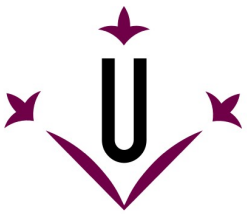

Universitat de Lleida

Escola Politècnica Superior
Enginyeria en Informàtica

Sistemes informàtics (Treball de final de carrera)

Code 2.0.

Gestió de pràctiques en núvol

Autora: Yolanda Vidal Falcó
Director: Juan Manuel Gimeno Illa
Juliol del 2013

Índex

Índex de figures

Capítol 1

Introducció

1.1 Context Actual

Des que al juliol de 1961 Leonard Kleinrock va publicar el primer document sobre la teoria de commutació de paquets, Internet ha tingut una evolució espectacular, modificant les conductes de les persones al treball, l'oci i el coneixement a nivell mundial.

Actualment, milers de persones tenen un accés fàcil i immediat a una extensa quantitat d'informació en línia, i és gràcies a la incorporació d'aquests usuaris a la xarxa que Internet proporciona moltes més possibilitats, com per exemple:

- Comunicació entre els usuaris: correu electrònic, missatgeria instantània, foros i notícies, blogs i fotologs.
- Oportunitats de negoci: E-business
- Oci i entreteniment: descàrregues (P2P), compres, jocs en línia.

Per tal d'aprofitar part dels recursos que ofereix Internet, va sorgir la primera versió del projecte "Code". "Code" és una aplicació web que va ser desenvolupada per l'entrega i avaluació d'exercicis de programació amb la finalitat de millorar la comunicació i intercanvi d'opinions entre alumnes i professors. Per aquest motiu, es va crear un entorn on, visualitzant el codi d'una activitat, tant alumne com professor podien realitzar comentaris sobre aquest. L'aplicació utilitza un entorn web per a que tothom en pugui fer ús des de qualsevol màquina connectada a Internet.

Tot i que aquesta primera versió de l'aplicació es va desenvolupar utilitzant eines innovadores al moment de la seva creació, han sorgit noves tecnologies i plataformes que poden ajudar al desenvolupament i el desplegament de l'aplicació en un entorn productiu. Aquest fet fa plantejar la realització d'una tasca molt habitual en informàtica; l'actualització d'una aplicació que ha quedat desfassada, per tal d'adaptar-la a la tecnologia del moment actual.

1.2 Estructura i Eines utilitzades per l'aplicació Code

L'entorn de desenvolupament de Code va ser Netbeans. Per crear l'estructura del projecte, es va crear com una nova Aplicació Web, seleccionant la versió de java J2EE 1.4 i marcant el

framework que es volia utilitzar, en aquest cas, Google Web Toolkit (prèvia instal·lació del plugin GWT4NB).

L'estructura inicial de directoris que va tenir el projecte Code, va ser la que es mostra en la figura 1.1.

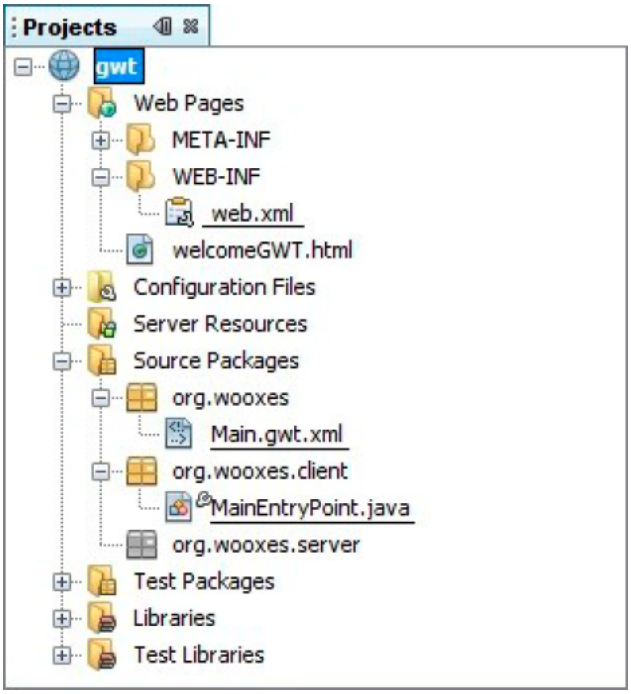

Figura 1.1: Estructura inicial projecte GWT Code

Un cop creada l'estructura, es va anar creant l'aplicació utilitzant el mecanisme RPC que incorpora Google Web Toolkit, ja que un dels aspectes més importants que té l'aplicació Code és la comunicació entre client i servidor, ja que freqüentment s'estan intercanviant informació.

A continuació es descriuen les principals eines que es van utilitzar pel desenvolupament de la primera versió de l'aplicació "Code":

- **Java** (1.4). És el llenguatge de programació base en el que s'ha programat aquesta aplicació. Va ser desenvolupat per Sun Microsystems a princips dels anys 90 i avui en dia és un dels més utilitzats. Les característiques principals són que és un llenguatge orientat a objectes (POO) i és multiplataforma, el que permet executar el mateix aplicatiu sobre diferents arquitectures i sistemes operatius.

- **GWT** (1.4.62). És un framework desenvolupat per Google que permet crear aplicacions web amb AJAX, programant amb Java. GWT s'encarrega de compilar i transformar el codi automàticament amb JavaScript, el qual pot ser interpretat per qualsevol naqvegador.

- **JavaScript**. És un llenguatge de programació interpretat, és a dir, no necessita ser compilat. La principal característica és que és el llenguatge per extendre les capacitats del navegador.

- **AJAX**. És una arquitectura que combina diferents tècniques:

  - Presentació amb XHTML + CSS
  - Interacció dinàmica utilitzant DOM
  - Peticions i respostes utilitzant JSON[8]
    JSON és un format d'intercanvi de dades que es basa en un subconjunt del llenguatge de programació JavaScript que no requereix l'ús de XML.
  - Recuperació de dades amb XMLHttpRequest
  - Unió de totes les tecnologies utilitzant JavaScript

- **XML** *(eXtensible Markup Language).* És un metallenguatge (especificat pel W3C) [1] amb el que definir llenguatges específics per tal de poder ser tractats amb eines estàndards.

Per la persistència de les dades de l'aplicació es van utilitzar dos medis diferents:

- Per la informació més consultada, com per exemple anuncis, assignatures, activitats i informació d'entregues, es va utilitzar el SGBD[2] **PostgreSQL**.
- Per altra banda, per la informació menys consultada, com per exemple els comentaris, es van utilitzar fitxers XML.

Finalment, el desplegament de l'aplicació es va realitzar sobre el servidor web **Apache Tomcat**.

1.3 Eines utilitzades per l'aplicació Code 2.0

1.3.1 Maven [9]

Durant el desenvolupament d'un projecte sempre hi ha diferents tasques a realitzar. La primera sol ser crear una estructura de directoris pel projecte amb el codi, les icones, els fitxers de configuració, les dades, etc. Després, altres tasques que es realitzen amb freqüència són compilar, generar la documentació, empaquetar l'aplicació, etc. En alguns casos, s'haurà de dependre d'algunes llibreries externes per facilitar la programació, com els drivers de la base de dades, JUnit per classes de test, log4j pels missatges de depuració, etc.

Avui en dia existeixen diferents eines de gestió de projectes que ajuden a realitzar aquestes tasques d'una manera més fàcil i simple. Algunes de les més representatives són:

- **Make** és una eina de generació de codi utilitzada en els sistemes operatius del tipus Unix/-Linux. S'utilitza per la creació de fitxers executables, per la seva instal·lació o desinstal·lació, etc.
- **Ant** és una eina utilitzada en la programació per la realització de tasques mecàniques, normalment durant la fase de compilació i construcció (build). És similar a Make però desenvolupada amb Java. Es basa en arxius de configuració XML i classes Java per la realització de les diferentes tasques, presentant-se així com una solució idònea per la programació multi-plataforma.

[1] World Wide Web Consortium: Consorci fundat al 1994 amb la finalitat de definir les especificacions pel desenvolupament de la tecnologia web

[2] Sistema de Gestió de Bases de Dades

Una eina més evolucionada és **Maven**. És una eina de software per la gestió i construcció de projectes Java creada per Jason Van Zyl l'any 2002. Consta d'un model de configuració simple, basat en un fitxer XML anomenat POM[3] en el que s'hi descriu el projecte a construir, les seves dependències, components externs i l'ordre de construcció dels elements.

Una de les principals característiques de Maven és la gran quantitat de tasques que realitza de forma automàtica si es segueix la metodologia de Maven, això comporta que per exemple, el fitxer pom.xml sigui molt més simple que un fitxer build.xml d'Ant. Maven facilita una sèrie de tasques relacionades amb la gestió de projectes:

1. Informació tècnica del projecte, com poden ser els repositoris dels artefactes, els llocs web o el sistema de control de versions.
2. Especificació de la ubicació de les fonts (fitxers .java).
3. Especificació de la ubicació dels recursos o fitxers de suport a incloure dins del classpath (fitxers XML, fitxers de propietats, etc).
4. Especificació de les llibreries jar a incloure dins del classpath del programa (Gestió de dependències).
5. Configuració del compilador de Java per compilar un tipus de JDK concret.
6. Composició dels diferents mòduls que construeixen el projecte.
7. Execució de test unitaris i/o d'integració.
8. Generació d'informes.
9. API per l'extenció de la funcionalitat a partir de plugins.

A nivell de construcció (build) de projectes Java, Maven proporciona un conjunts de tasques comunes que es poden realitzar en un projecte:

1. Compilar les fonts: compile.
2. Netejar el directori de classes generat: clean.
3. Copiar els recursos a la seva ubicació final.
4. Empaquetar/Comprimir les classes en algun tipus de paquet Java (.jar, .war, .ear, .rar): package.
5. Executar els tests.
6. Generar els Javadocs.
7. Desplegar l'aplicació al servidor d'aplicacions: deploy.

En definitva, els **objectius** de Maven són:

[3] Project Object Model

- Facilitar el procés de construcció.
- Proporcionar un sistema de construcció unificat (pom.xml).
- Proporcionar informació sobre el projecte.
- Facilitar la migració a noves característiques o funcionalitats.

1.3.2 Control de versions

El control de versions és el procés d'administrar diferents versions d'un conjunt d'informació. Avui en dia administrar manualment moltes versions de fitxers és una tasca propensa a errors, encara que fa temps que existeixen eines que ajuden amb aquest procés.

Es pot establir dos tipus de control de versions:

1. **Centralitzats**

   Es una estructura que estableix un node central per guardar tot el codi que està a disposició de tots els usuaris. S'emmagatzema en un únic directori, normalment allotjat a un servidor, i tots els programadors que treballen sobre el codi han de demanar al servidor una còpia per treballar de forma local. Un cop realitzades les modificacions sobre la seva còpia local, s'exporten els canvis al servidor, canviant les diferències que poden existir entre la copia local i la del servidor.

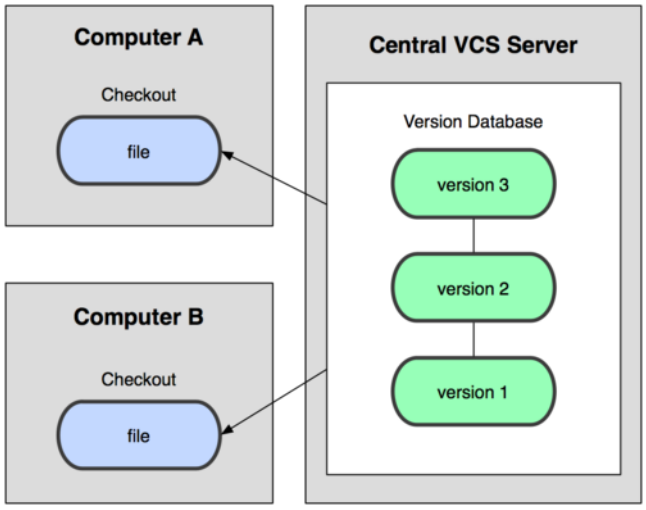

Figura 1.2: Diagrama de control de versions centralitzat

Característiques:

- És un sistema més senzill d'utilitzar.
- Es té un control total sobre les versions.
- Limitacions amb l'accés.
- Menys conflictes a resoldre quan hi ha algun tipus de problema.

Uns exemples d'aquest sistema són els següents:

- **CVS**[4]: És l'eina més utilitzada pel control de versions. Va sorgir al 1990 i durant molt de temps ha estat el gestor de versions de molts projectes de codi obert. Consta d'una arquitectura centralitzada client/servidor.
- **Subversion**: És una eina de control de versions molt popular, desenvolupada a l'any 2000 per CollabNet i preparada per reemplaçar a CVS. S'utilitza en projectes de codi obert (Apache, PHP, DJango, Mono) entre d'altres. Té una arquitectura centralitzada del tipus client/servidor.

2. **Distribuïts**

   En aquest tipus de sistema no hi ha un servidor que contingui una còpia del repositori, sinó que cada usuari té el seu propi repositori. Tots els canvis són locals i un cop s'han realitzat les modificacions pertinents, s'estableix una sincronització amb els repositoris de cada un dels usuaris.

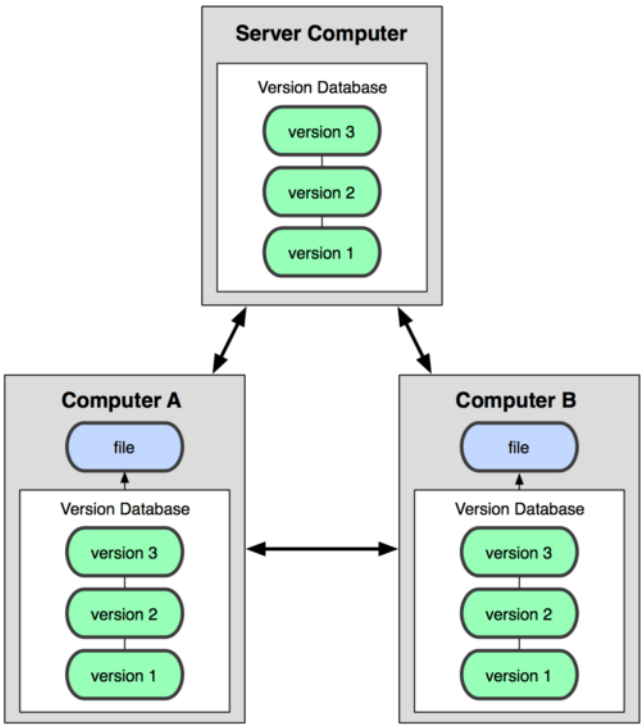

Figura 1.3: Diagrama de control de versions distribuïts

Característiques:

- Es pot treballar de forma local, això comporta una millora de velocitat ja que no es necessita una red per connectar amb el servidor.
- Les operacions locals són més ràpides.
- No es necessari demanar permís per participar en projectes.
- No depèn d'un sola màquina física.

[4]Concurrent Versions System

- Es pot seguir tenint un control centralitzat del projecte.

Uns exemples són:

- **Mercurial** [10]: És un sistema de control de versions multiplataforma. Va sorgir l'any 2005 a càrrec de Matt Mackall. Avui en dia es utilitzat per Mozilla, OpenSolaris, OpenOffice, Netbeans i GO (el llenguatge de programació de Google).
- **GIT** [2]: És una eina de control de versions distribuïda dissenyada per Linus Torvals l'any 2005 i desenvolupada per administrar l'arbre del kerner de Linux. El seu disseny es va basar amb Monotone i BitKeeper (eines de software pel control de versions).
- **Bazaar** [1]: És un sistema de control de versions distribuït i desenvolupat amb Python. Ajuda a dur a terme la història del projecte en el temps i col·laborar fàcilment amb altres programadors adaptant-se a les seves necessitats. Bazaar està promocionat per Canonical [5].

Existeixen vàries raons per les que es recomanable utilitzar una eina automàtica de control de versions en un projecte:

- Controla la història i l'evolució del projecte per evitar realitzar feina manualment. Per cada canvi es registra qui l'ha fet, per què, quan i de quin canvi es tracta.
- Treballar simultàniament, facilita la col·laboració entre els membres d'un equip de treball.
- Ajuda a recuperar versions anteriors. Si s'aplica un canvi que posteriormet s'identifica com un error, es pot aplicar una versió prèvia.
- Permet controlar les diferències entre les múltiples versions del projecte.

1.3.2.1 Mercurial

Mercurial està implementat principalment amb el llenguatge de programació Python. En un principi va ser escrit per funcionar sobre Linux, però actualment s'ha adaptat per Windows, Mac OS X i per la majoria de sistemes operatius tipus Unix.

És un programa per línia de comandes, així totes les operacions s'invoquen com arguments al programa principal.

Les principals avantatges d'aquest sistema de control de versions són:

- Es fàcil d'aprendre i utilitzar.
- És lleuger.
- Escala de forma excel·lent.
- Es fàcil d'acondicionar.
- Té un gran rendiment.

[5] Empresa privada amb l'objectiu de promocionar projectes relacionats amb el software lliure

- Consta d'un desenvolupament distribuït, sense necessitat d'un servidor.
- Té una gestió robusta de fitxers, tant de text com binaris.
- Capacitats avançades d'integració.

1.4 Tecnologies utilitzades en l'aplicació

Un dels objectius d'aquest projecte és actualitzar les versions de les tecnologies utilitzades en l'aplicació. A continuació es descriuen les tecnologies utilitzades, així com les diferències entre les versions d'aquestes en l'aplicació actual i Code.

1.4.1 Java [6]

Java és una tecnologia orientada al desenvolupament de software amb la qual podem realitzar qualsevol tipus de programa. Està basat bàsicament per dos elements: el llenguatge Java i la seva plataforma (la màquina virtual de Java: *Java Virtual Machine*). Actualment el llenguatge Java es un dels més utilitzats. Va ser desenvolupat per Sun Mycrosystems a principis dels anys 90. La principal característica es que és un llenguatge independent de la plataforma, es a dir, que podem executar el mateix aplicatiu sobre diferents arquitectures i sistemes operatius.

El llenguatge es va crear amb cinc objectius principals:

1. Hauria d'utilitzar la metodologia de la programació orientada a objectes.
2. Hauria de permetre l'execució d'un mateix programa en varis sistemes operatius.
3. Hauria d'incloure por defecte suport de treball en xarxa.
4. Hauria de ser dissenyat per executar codi en sistemes remots de forma segura.
5. Hauria de ser fàcil d'utilitzar i adaptar el millor d'altres llenguatges orientats a objectes, com C++.

Java ha experimentat nombrosos canvis desde la seva primera versió, 1.0, així com un gran increment en el nombre de classes i paquets que composen la llibreria estàndard.

Java 1.4 (2002) - Anomenat Merlin

- Millora del rendiment.
- Nova API d'entrada/sortida de baix nivell (NIO, NEW I/O).
- Classes per treballar amb Collections.
- Millores de seguretat: suport per la criptografia mitjançant Java Cryptography Extension (JCE), la inclusió de Java Secure Socket Extension (JSSE) i Java Authentication and Authorization Service (JAAS).

Java 1.5 (2004) - Anomenat Tiger o Java 5.0

En aquesta versió s'inclou com a principals novetats:

- Tipus genèrics [11].

  Els tipus genèrics permeten definir dins d'una classe un tipus de paràmetre que assumeix el tipus de dades amb el qual es desitja treballar.

- Enumerats.
- Bucles simplificats.
- Funció printf.
- Funcions amb número de paràmetres variables.
- Metadades a les classes i mètodes.

Java 1.6 (2006) - Anomenat Mustang

- Sun va canviar el nom de "J2SE" per Java SE.
- Inclou un nou marc de treball i APIs que fan possibles la combinació de Java amb llenguatges dinàmics com PHP, Python, Ruby i JavaScript.
- Inclou el motor Rhino, de Mozilla, una implementació de JavaScript amb Java.
- Inclou un client complert de Serveis Web i suporta les últimes especificacions per aquest.
- Millores a l'interfície gràfica i en el rendiment.

Java 1.7 (2011) - Anomenat Dolphin

- Gestió automàtica de recursos.
- Inferència de tipus millorada per a creació genèrica d'instàncies (operador diamant < >).
- Barra baixa en literals numèrics "_".
- Cadenes per la clàusula switch.
- Binaris literals.
- Simplificació de la invocació de mètodes amb arguments variables.

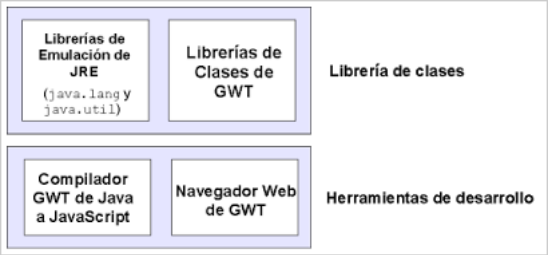

Figura 1.4: Arquitectura de GWT

1.4.2 Google Web Toolkit [4]

Google Web Toolkit (GWT) és un framework de desenvolupament de codi obert, creat per Google que permet desenvolupar i depurar aplicacions AJAX, utilitzant el llenguatge de programació Java. Un cop finalitzada l'aplicació, GWT compila i tradueix el programa a JavaScript i HTML compatible amb qualsevol navegador web.

Un dels avantatges de Google Web Toolkit es que el model de programació web que utilitza (servidor Java; client JavaScript), és similar al model de programació d'escriptori: Swing, ja que tant un com l'altre utilitzen Widgets, panels, events i es desenvolupen amb el llenguatge Java.

Les principals característiques de GWT es detallen a continuació:

- Conté un conjunt de components per la interfície gràfica d'usuaris dinàmics (Widgets).
- Utilitza un RPC[6] fàcil, ja que ens podem comunicar des del navegador que inicia l'aplicació amb el servidor web, definint classes de Java serialitzables per les peticions i respostes.
- Permet guardar un estat concret de l'aplicació a l'historial.
- Les aplicacions són compatibles amb la majoria de navegadors com per exemple FireFox, Internet Explorer, Mozilla, Safari o Opera.
- S'integra Junit a GWT per poder provar les aplicacions i depurar-les al navegador mentre s'estan executant.
- Permet desenvolupar aplicacions i biblioteques d'internacionalització de forma ràpida, fàcil i senzilla.
- Es pot inserir JavaScript al codi de l'aplicació utilitzant la interfície JSNI[7], si les llibreries de classes de GWT no són suficients.
- És un projecte de codi obert sota la llicència Apache 2.0.

L'arquitectura de Google Web Toolkit té quatre components principals: un compilador Java-a-JavaScript, un navegador web "hosted", i dos llibreries de classes, tal i com es mostra a la figura 1.4:

Els components que composen l'arquitectura són:

[6]Remote Procedure Call: És un mecanisme de computació distribuïda que permet a un programa client abstraure's d'alguns aspectes de baix nivell per tal d'executar codi en una màquina remota.

[7]JavaScript Native Interface

- **Compilador GWT Java-a-JavScript.**

  El Compular GWT tradueix del llenguatge de programació Java a JavaScript. El compilador s'utilitza quan es necessita executar l'aplicació en un navegador web.

- **Navegador web "hosted"**

  El navegador "hosted" que implementa GWT permet executar l'aplicació des d'una màquina virtual de Java, sense traduir-la a JavaScript, gràcies a l'ús d'un controlador especial que utilitza amb el navegador (Internet Explorer sobre Windows o Firefox sobre Linux).

- **Emulació llibreries JRE**

  GWT conté implementacions en JavaScript de les classes més utilitzades en Java, incloent la major part del paquet java.lang i java.util. La resta de biblioteques estàndard de Java no són suportades de forma nativa. Per veure més en detall, consultar l'annex "Suport de Java en GWT".

- **Biblioteca de classes d'interfície d'usuari**

  Són un conjunt d'interfícies i classes personalitzades que permeten crear "widgets" per al navegador, tal com botons, caixes de text, imatges, text... necessaris per a la creació d'aplicacions GWT.

En el projecte inicial Code es va utilitzar la versió 1.4, però la nova versió d'aquest framework aporta noves funcionalitats que fan que sigui la més utilitzada actualment.

GWT 1.4

- S'han afegit nous elements (com *RichTextArea*, *HorizontalSplitPanel*, *VerticalSplitPanel*, *SuggestBox*, etc), que permeten crear aplicacions avançades més fàcilment.
- El component *ImageBundle* consolida automàticament múltiples imatges en una única petició HTTP.
- Els components *NumberFormat* i *DateTimeFormat* faciliten la internacionalització.
- Un nou subsistema de rendiment basat en JUnit permet medir i comparar la velocitat de parts del codi per realitzar testos unitaris fàcilment.
- Tot el codi està sota la llicencia "Apache 2.0" i la documentació web sota la "Attribution de Creative Commons".

GWT 1.5

- Suport per Java 5.

  S'agrega suport per annotacions, genèrics, enumeracions i gran quantitat de paràmetres variables als mètodes.

- Optimitzacions al compilador.

  S'ha millorat la performance de les aplicacions compilades. Les aplicacions existents s'hauran de recompilar per beneficiar-se de les noves millores.

- Millores a JavaScript.

  S'ha creat una nova API pel DOM amb millores a la performance.

- Estils visuals predeterminats

  S'afegeixen varis estils CSS per les presentacions per poder desenvolupar una aplicació atractiva.

GWT 1.6

- Nova estructura del projecte.

  S'introdueix una nova estructura de projecte basada en el fitxer web J2EE estàndard (format war). Això comporta que sigui molt més fàcil agregar un mòdul de GWT en una aplicació web existent.

- Sistema d'events

  El sistema d'events s'ha actualitzat per a que sigui compatible amb les normes de Java, per tant, és més fàcil d'utilitzar i més fàcil d'escriure els propis widgets. S'han creat noves classes i nous mètodes, com per exemple les classes EventHandler i HandlerManager.

- Nous reproductors.

  Els nous widgets DatePicker i Datebox permeten als usuaris seleccionar una data d'un calendari.

GWT 1.7

- Suport per Explorer 8, Firefox 3.5 i Safari 4.

GWT 2.0

- Development Mode

  Substitueix el "hosted mode". A partir d'aquesta versió es pot desenvolupar i depurar el codi que s'executa al Safari, Firefox, Internet Explorer i Chrome. El codi font no té perquè estar al navegador web, es a dir, es pot desenvolupar i depurar el codi en un sistema operatiu, mentre s'executa l'aplicació en un sistema separat.

- Anàlisi de rendiment: Speed Tracer

  És una nova eina que ajuda a identificar i solucionar problemes de rendiment a les aplicacions web. Conté un traçador de velocitat capaç de tenir una idea del temps que es s'inverteix en una aplicació.

- Code Splitting

  Són particions de codi guiades pel programador. El codi de JavaScript es descarrega sota la petició del programador: GWT.runAsync.

- Optimitzacions del compilador per fer que el compilat de JavaScript sigui més petit i més ràpid.

- Interfícies d'usuari

  Amb UiBinder, GWT permet crear interfícies d'usuari mitjançant declaracions amb XML enlloc de programar-ho, d'aquesta manera serà més fàcil de llegir, mantenir i més ràpid de desenvolupar.

- Panels de disseny
- Construcció de recursos via ClientBundle
- HTMLUnit per realitzar proves.
- Depuració multi-navegador.

  A partir de GWT 2.0 es pot depurar el codi des del navegador. Es pot executar l'aplicació al navegador i utilitzar eines com *Firebug* mentre depurem des de l'Eclipse, Netbeans, etc.

GWT 2.1

- Cell Widgets

  Aquests widgets han estat dissenyats per manejar i visualitzar grans conjunts de dades ràpidament. Pot acceptar dades de qualsevol font de dades. El model de dades s'encarrega de les actualitzacions asíncrones. Quan es canvia les dades, la vista s'actualitza automàticament.

- MVP Framework (Model - Vista - Presentador)

  Les característiques d'aquest model són:

  - La Vista no coneix el model
  - El Presentador és independent de la interfície d'usuari
  - Es poden realitzar proves sobre la Vista i el Presentador

- Request Factory

  És l'alternativa a GWT-RPC per la creació de serveis orientats a dades: validacions, persistència, etc.

  Implementa el seu propi protolocol per l'intercanvi de dades entre el client i el servidor,

- Editors

  Permet de manera fàcil crear widgets per editar objectes de domini.

- Server-side Speed Traces

  Al desplegar una aplicació amb Google App Engine o SpringSource TC Server, es pot utilitzar l'eina Speed Tracer per analitzar el rendiment.

- Logging

  El framework de logging emula java.util.logging. Permet compartir el codi de logging entre el client i el servidor. Es configura al fitxer gwt.xml.

- Safe HTML

- Integració amb eines de desenvolupament SpringSource

  Spring Roo és una eina de desenvolupament d'aplicacions Java. Es caracteritza per un desenvolupament ràpid i fàcil per obtenir resultats immediats.

GWT 2.2

- Integració de Drag and Drop UI Designer

  Google Plugin per Eclipse 2.2 integra un lleugera versió de GWT Designer, un potent dissenyador d'interfícies d'usuari que permet que sigui molt més fàcil i ràpid construir una interfície. Amb aquesta possibilitat, s'inclou un suport per UIBinder i Cell Widgets.

- Actualitzacions del widget CellTable.
- S'ha integrat nous events: Touchstart, Touchmove, Touchend i Touchcancel.
- Desapareix el suport amb Java 5.

GWT 2.3

- S'ha afegit suport per Internet Explorer 9.
- Noves funcionalitats pel plugin de Google per Eclipse:
  - Integració de l'API de Google
  - Importació de projectes de Google Project Hosting
  - Inici de sessió únic per tenir accés a Project Hosting i App Engine
- API d'emmagatzemament local.

GWT 2.4

En aquesta nova versió s'han introduït importants millores que faciliten el desenvolupament i milloren el rendiment.

- Injecció de CSS en funció del navegador (StyleInjector + CssResource).

  La CSS no es interpretada igual en tots els navegadors. En aquesta versió hi ha diferents CSS per a diferents navegadors i només es descarrega quan es necessari.

- Descàrregues de l'aplicació incremental.
- Inspecció del rendiment.
- App Engine connectat a Android.
- Millores en el dissenyador de GWT.
- Suport al App MarketPlace

Figura 1.5: Cloud Computing

1.4.3 Google App Engine [3]

Cloud Computing és una forma de computació compartida on els requeriments informàtics es presten com un servei. Normalment aquests serveis estan en un centre de dades (Cloud o núvol) on s'hi permet l'accés, sense la necessitat d'una infraestructura definida (còmput, capacitat d'emmagatzematge, etc.) o uns coneixements previs per part de l'usuari.

Un exemple de Cloud Computing es Google App Engine. Es defineix com un servei que permet executar als servidors de Google aplicacions web fàcils de crear, mantenir i actualitzar a mesura que van augmentant les necessitats de còmput, d'emmagatzemantge de dades i ample de banda. Aquest servei va ser llançat a la primavera del 2008 com la "*plataforma de la nube*" en la que qualsevol persona pot allotjar el codi a la infraestructura de Google.

La principal característica de **GAE** es l'escalabilitat automàtica de les aplicacions. Una aplicació *"escala automàticament*" quan després de la implementació amb Google App Engine, aquesta pot servir a molts usuaris connectats simultàniament sense degradar el seu rendiment. A mesura de que més usuaris accedeixen a la aplicació, GAE atorga i administra més recursos sense que l'aplicació s'hagi de preocupar o saber alguna cosa dels recursos que utilitza. A més a més, Google App Engine ofereix un conjunt d'eines (API's) que incrementen la productivitat en el desenvolupament.

Les aplicacions es poden executar en dos entorns d'execució diferents, on cada un proporciona protocols estàndards i tecnologies comunes pel desenvolupament d'aplicacions: la plataforma Java i l'entorn Python.

No obstant, desenvolupar una aplicació amb Google App Engine obliga al programador a adequar-se a les limitacions que aquesta plataforma imposa:

- Les aplicacions solament tenen permissos de lectura als fitxers del sistema de ftixers. Per emmagatzemar dades i fitxers en mode de lectura i escriptura es necessari utilitzar un sistema de fitxers virtual sobre el DataStore.
- No totes les llibreries i frameworks poden utilitzar-se pels llenguatges suportats pel SDK de GAE, moltes d'aquestes limitacions es deuen a la incapacitat de poder escriure sobre el sistem d'arxius de Google (GFS).

- No existeix una base de dades relacional, ja que es molt complicat d'escalar un sistema distribuït.
- No és portable, això signifca que si es vol canviar de proveïdor, s'haurà de re-escriure part o tota l'aplicació, ja que les API de Google App Engine no tenen el codi obert, i són específiques per la insfraestructura de Google.
- Alguns serveis i eines tenen limitacions de temps de resposta, capacitat de memòria de les peticions, quotes diàries, entre d'altres.

Google App Engine, incorpora serveis que faciliten el desenvolupament:

- **Datastore**. Servei robust i escalable que permet persistir la informació. És un emmagatzemament de dades no relacional.
- **Memcache**. Servei d'emmagatzemament volàtil, ideal per persistir la informació que s'accedeix amb molta freqüència i no està sotmès a molts canvis.
- **Mail**. Servei que permet enviar i rebre correus electrònics des de l'aplicació.
- **URL Fetch**. Permet accedir a serveis externs a través de peticions, utilitzant HTPP i HTTPS.
- **Task Queue**. Servei que permet realitzar treballs fora d'una petició realitzada per un usuari. Permet a l'usuari organitzar el treball amb diverses tasques.
- **Imatges**. Ofereix la possibilitat de manipular imatges utitlizant un servei dedicat a les imatges. Permet canviar la dimensió, girar, reajustar imatges, entre d'altres.
- **Blobstore**. Permet a l'aplicació servir objectes, anomenats Blobs, que són més grans que els objectes permesos al datastore.
- **User**. API per autenticar els usuaris que tinguin compte de Google, comptes d'un domini propi utilitzat a Google Apps.
- **Channel**. Servei de connexió persistent entre l'aplicació i els servidors de Google, utilitzant funcions de consulta, ordre i transaccions.
- Un entorn de desenvolupament local que simula Google App Engine al teu equip.

1.5 Objectius

Els objectius principals del projecte són:

- Flexibilitzar i dinamitzar la construcció de l'aplicació, facilitant als programadors tant l'etapa de codificació com les possibles modificacions que es puguin dur a terme sobre l'aplicació.

  Amb aquest objectiu s'ha integrat "Code" en una eina de software per la gestió i construcció de projectes Java com és Maven.

- Incorporar al projecte les millores implementades en les noves versions de les llibreries GWT i Java.
- Escalabilitzar l'aplicació a través de la tecnologia Cloud Computing, concretament amb la infraestructura Google App Engine.

Capítol 2

Desenvolupament de l'aplicació

La migració de l'aplicació original a la nova implementació realitzada en aquest projecte (Code 2.0) s'ha dut a terme seguint una metodologia iterativa. En la etapa de inicialització, s'ha creat una llista amb les submigracions que s'han de dur a terme en cada iteració i que s'expliquen a continuació.

En la primera i segona iteració s'ha integrat les noves eines aportades en aquest projecte: Mercurial i Maven. Tal i com s'ha explicat en les seccions 1.3.2.1 i 1.3.1 respectivament, Mercurial permet un control automàtic de les versions de l'aplicació i Maven facilita una sèrie de tasques relacionades amb la gestió del projecte.

En la tercera iteració s'ha actualitzat la versió Java Runtime Environment de la versió 4 a la 6 i s'ha recompilat tota la plataforma en aquesta nova versió.

Un cop ja creada l'estructura del projecte amb la integració de les dos noves eines i l'actualització de Java, la quarta iteració ha consistit en l'actualització del SDK Google Web Toolkit a la versió 2.4.0. Al tractar-se d'una actualització de versió base (de 1.4.6 a 2.4.0) s'han hagut de realitzar modificacions importants al codi del projecte.

En la cinquena iteració s'ha ampliat la funcionalitat de la plataforma amb la llibreria *Google App Engine* amb l'objectiu de poder executar l'aplicació sobre una arquitectura Cloud Computing com la que ofereix Google.

Finalment s'ha allotjat l'aplicació a un servidor de Google per la seva distribució.

2.1 Control de versió: Mercurial

El primer pas que hem de realitzar es intal·lar Mercurial pel nostre sistema operatiu, en aquest cas, per Mac OS X. Si nó realitzem aquest pas, el plugin d'Eclipse no podrà utilitzar Mercurial perquè no el trobarà al sistema.

Descarreguem l'instal·lador de la pàgina oficial: http://mercurial.selenic.com/. Un cop descarregat, seguim els passos de l'instal·lador.

Per comprovar si la instal·lació s'ha realitzat correctament, es pot utilitzar la comanda *hg version*, tal i com es mostra a la figura 2.1

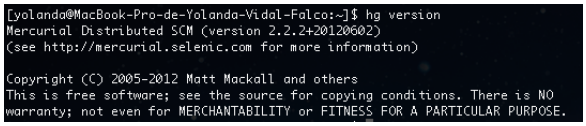

Figura 2.1: Comprovació Mercurial

Mercurial funciona al voltant d'un *repositori*. Un repositori és un directori que conté tots el codi font que es vol mantenir juntament amb una història d'aquests fitxers.

Els passos a seguir per tal d'utilitzar un repositori amb mercurial sobre la plataforma Eclipse són els que es detallen a continuació:

1. Instal·lem el plugin HgEclipse. Per instal·lar-ho, seleccionarem l'opció **Help > Install New Software** i introduirem el repositori al camp **Work with**: *http://cbes.javaforge.com/update*.

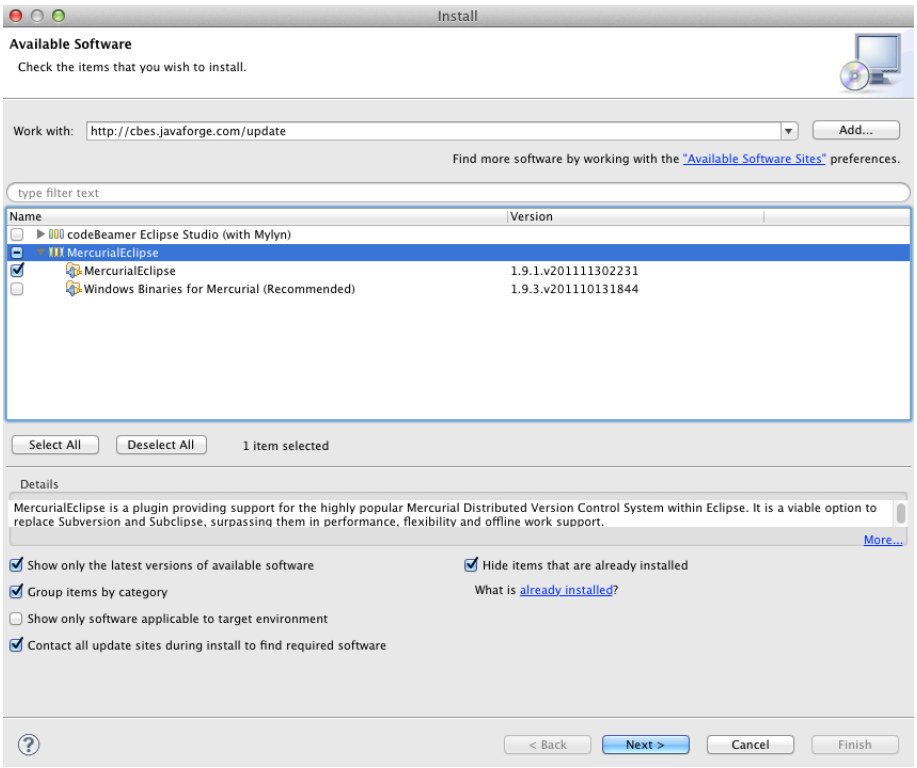

Figura 2.2: Repositori Mercurial

2. A continuació, premem el botó Next, es descarregà i a continuació s'instal·larà el plugin.

3. Un cop instal·lat, es crea el repositori Mercurial al projecte, d'aquesta manera es podrà gestionar les versions del codi font. El primer pas es compartir el projecte fent botó dret damunt del projecte, seleccionant l'opció **Team > Share Project** i el tipus de repositori: Mercurial, tal i com es mostra a la figura 2.3

4. A continuació, ens mostrarà on es crea el repositori. Un cop creat, apareixeran noves icones a les carpetes i fitxers del projecte i la paraula [new] indicant que el repositori es nou però que encara no s'ha realitzat el primer commit (figura 2.4).

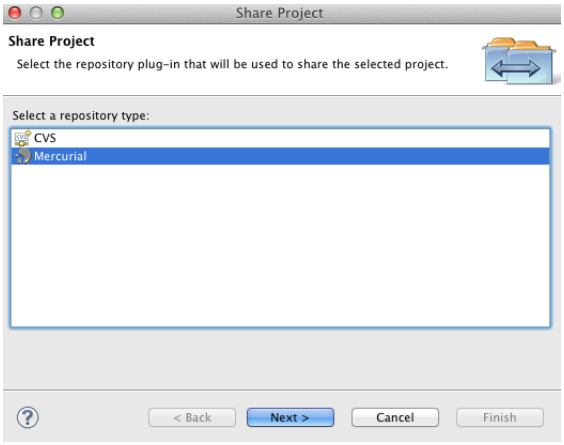

Figura 2.3: Share Projecte

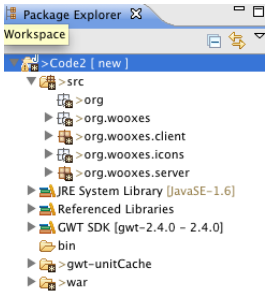

Figura 2.4: Repositori després del init

5. Per afegir fitxers al repositori, seleccionem l'opció **Team > Share Project > Add**. Un cop afegits, les icones dels fitxers canvien, mostrant un "+" de color blau.

6. En aquests moments els fitxers estan controlats pel repositori però no s'han pujat els canvis. Per fer-ho, seleccionem l'opció **Team > Share Project > Commit** i afegim un missatge de control, tal i com es mostra a la figura 2.5.

7. A partir d'ara, qualsevol canvi que es realitzi en un fitxer, canviarà la icona mostrant un "*" de color negre i llavors podrem fer un commit per pujar els canvis.

2.2 Integració de l'aplicació amb Maven

Maven és una eina de software per la gestió i construcció de projectes Java. Utilitza un fitxer de configuració anomenat pom.xml. Pot semblar que Maven és similar a Ant però l'enfocament és molt diferent. Ant vol automatitzar tasques amb dependències; Maven vol gestionar el control de projecte. Ant és imperatiu; Maven és descriptiu (es a dir, Ant diu què fer, i Maven descriu l'estructura). Una de les característiques més importants és la seva actualització en línia mitjançant repositoris, es a dir, és capaç de descarregar-se noves actualitzacions de les llibreries de les que depen el projecte.

Per integrar Maven a l'aplicació Code2.0, utilitzarem un plugin de Maven per Eclipse, m2e, ja que permet treballar de manera més fàcil amb projectes Java.

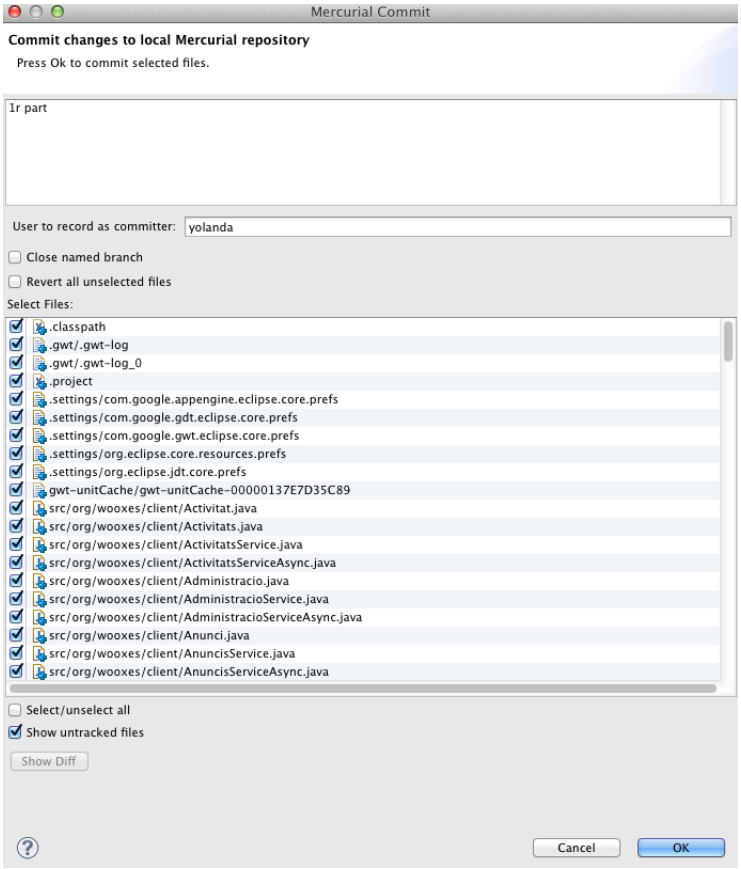

Figura 2.5: Commit

m2e es un plugin que permet a l'usuari interactuar amb els formats de múltiples repositoris, facilita l'edició del fitxer de configuració pom.xml i simplifica el consum dels artefactes de Java, ja estiguin allotjats als repositoris de codi obert o als propis repositoris de Maven.

Per instal·lar el plugin de Maven seguirem els passos que s'han seguit en la secció 2.1.1 afegint la URL *http://m2eclipse.sonatype.org/sites/m2e*.

Amb la integració de Maven a l'aplicació, suposa un gran canvi tant en els fitxers de configuració com en l'estructura dels directoris d'aquesta. Per aquest motiu, s'ha optat per crear un projecte des de zero i afegir les classes de l'antiga aplicació a la nova estructura. Per fer-ho s'ha executat la següent comanda de Maven: *./webAppCreator -maven -out Code2 org.wooxes*.

L'estructura de directoris que ens queda després de crear el projecte i afegir totes les classes a la carpeta corresponent es la que s'aprecia a la figura 2.6.

L'estructura d'un projecte Maven es una de les convencions de Maven, es a dir, en tots els projectes Maven trobarem la mateixa estructura de directoris. Aquesta anotació es important ja que si coneixen l'estructura de directoris d'un projecte, es coneix la de qualsevol projecte que utilitzi Maven.

Dins de la carpeta d'un projecte Maven tindrà la carpeta **src/**, aquesta conté tot el codi font que s'escriu per l'aplicació. src/ té dos subdirectoris **main/** (codi font principal) i **test/** (codi font de proves: JUnit). Cadascuna té una carpeta java que es on van les classes Java d'aquesta categoria

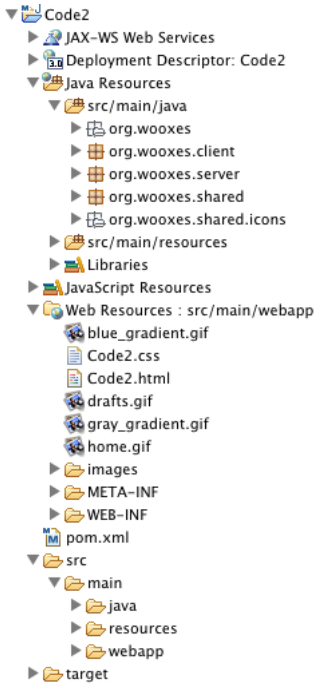

Figura 2.6: Estructura de l'aplicació

junt amb els seus paquets. Aquests són els directoris principals, però la convenció de Maven té més directoris:

- **pom.xml** (Configuració del projecte)
- **src/** (Codi font)
  - **main/** (Codi font principal)
    - **java/** (Codi font java)
      - <packages>
    - **resources/** (Recursos i fitxers de configuració a incloure al classpath)
    - **webapp/** (Si l'aplicació es web, aquí s'ubicarà els recursos web (imatges, css, html, etc.)
      - WEB-INF
  - **target/** (Directori que s'autogenera a l'invocar qualsevol de les accions de construcció de Maven, conté els fitxers generats: classes, jar, etc)

Per executar una aplicació amb Maven, es fa servir la terminologia de Goal. Un **goal** es un objectiu que indica que es vol i que s'especifica com un argument en una línia de comandes o a través del plugin instal·lat a l'Eclipse. Per aconseguir-ho Maven executa vàries tasques.

Els goals més utilitzats són els que s'especifiquen a continuació:

- **mvn clean:** Neteja el directo target del projecte.

- **mvn compile:** Compila el codi font del projecte.
- **mvn package:** Empaqueta el projecte en un arxiu jar, war, etc. depenent de la configuració del projecte.
- **mvn install:** Instal·la el projecte al repositori local.
- **mvn deploy:** Desplega l'aplicació en un repositori remot.

2.3 Actualització de Java 4 a Java 6 [11]

Una de les millores que va aportar Java 1.5 (i en conseqüència les següents versions) han estat els anomenats *Genèrics*. Un Genèric permet que un tipus pugui ser utilitzat com a paràmetre en la definició d'un mètode o d'una classe. S'utilitzen especialment per implementar tipus abstractes de dades: piles, cues i altres que permeten emmagatzemar diferents tipus d'elements segon siguin instanciats en temps de compilació.

Fins la versió 1.4 de Java, la genericitat s'aconseguia declarant una llista que acceptés objectes de tipus Object. Donat que en Java, totes les classes hereden d'Object, la classe Llista (List) pot emmagatzemar qualsevol tipus d'element mitjançant una substitució polimòrfica. Al recuperar els elemets de la llista, era necessari afegir els càstings pertinents depenent del tipus de dades que s'hagués introduit.

A continuació es detallen alguns problemes que es podien observar al treballar en aquesta versió:

- El programador havia de recordar el tipus d'elements que hi havia emmagatzemat a la llista i realitzar el càsting al tipus apropiat quan s'havia d'extreure de la llista.
- No es comprovava el tipus en temps de compilació. Pel compilador els elements de totes les llistes eren de tipus Object, per aquest motiu no hi havia comprovació.
- Era necessaria la comprovació explícita de tipus en temps d'execució. Al no diferenciar els tipus en temps de compilació es feia necessaria una comprovació per poder detectar possibles errors.
- Aparició d'excepcions (ClassCastException) en temps d'execució. A l'aparèixer en temps d'execució es feia molt més dificil l'eliminació dels errors de càsting.
- Es permetia la existència de classes contenedores amb objectes heterogenis que ocacionaven problemes a l'intentar recuperar els elements de les llistes.

A la versió 5 apareixen els anomenats *Genèrics* que solventen aquests tipus de problemes. S'utilitzen especialment per implementar tipus abstractes de dades: piles, cues y altres que permeten emmagatzemar diferents tipus d'elements segons estiguin instanciats en temps de compilació.

Al Juliol del 2011 apareix la versió 7 de Java (Dolphin) amb noves millores però amb l'inconvenient de que no es compatible amb Google App Engine. Quan l'aplicació Java s'executa a App Engine, ho fa a través de la màquina virtual de Java (JVM) 6 i de les biblioteques estàndars. Segons Google App Engine, l'ideal es utilitzar Java 6 per compilar i per provar l'aplicació, i assegurar de que el servidor local es comporta de forma similiar a App Engine.

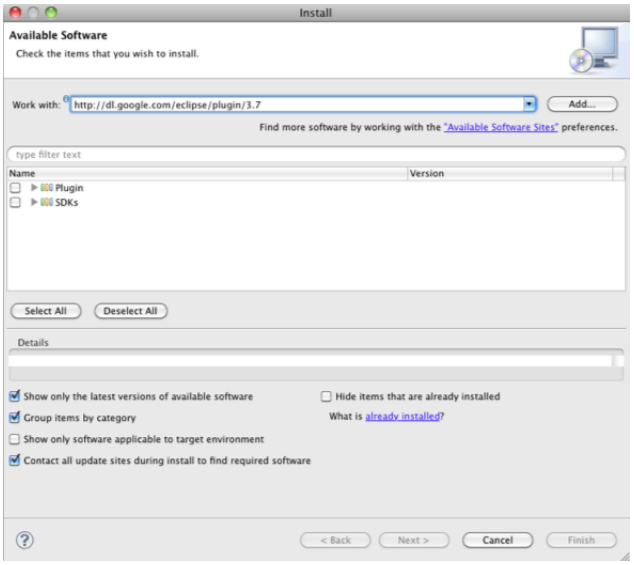

Figura 2.7: URL

2.4 Actualització de Google Web Toolkit

2.4.1 Instal·lació del plugin de Google per Eclipse

El plugin Google per Eclipse (GPE) és un conjunt d'eines de desenvolupament de sofware que permet als desenvolupadors de Java a dissenyar amb rapidesa, crear, optimitzar i distribuir aplicacions basades en un núvol. A més a més suporta diverses utilitats per la creació d'aplicacions web que utilitzen la plataforma i servidors de Google, a través de Google App Engine.

A continuació s'explica el procediment per instal·lar el plugin de Google per Eclipse, i opcionalment el SDKs del Google Web Toolkit i Google App Engine.

1. Un cop s'ha iniciat l'Eclipse, seleccionem l'opció del menú **Help > Install New Sofware.** A la finestra que apareix, introduïm al camp **Work with** la següent URL del lloc d'actualització: *http://dl.google.com/eclipse/plugin/3.7,* tal i com es mostra a la figura 2.7

2. La finestra central mostrarà els plugins i els SDKs disponibles. D'aquests seleccionem els components següents: el complement de Google per Eclipse, el SDK de Google Web Toolkit i Google App Engine (figura 2.8).

3. A continuació, revisem les característiques dels elements que volem instal·lar (figura 2.9).

4. Llegim i acceptem els acords de la llicència per començar la instal·lació. A continuació, s'instal·laran les dependències i els components elegits, tal i com es mostra a la figura 2.10.

5. Finalment, el sistema preguntarà si desitgem reiniciar l'aplicació. Fem clic en Reiniciar per aplicar les configuracions (figura 2.11).

Un cop configurat, a la barra del menú ens apareix una nova icona amb diferents opcions per poder treballar amb el plugin Google Web Toolkit (figura 2.12).

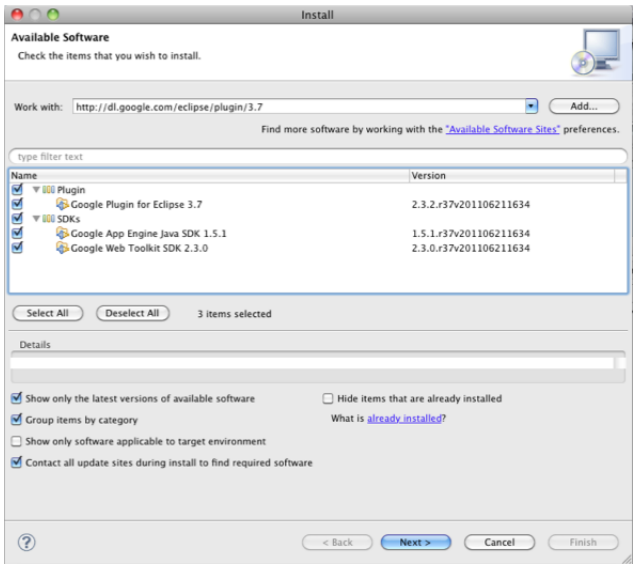

Figura 2.8: Plugins i SDK

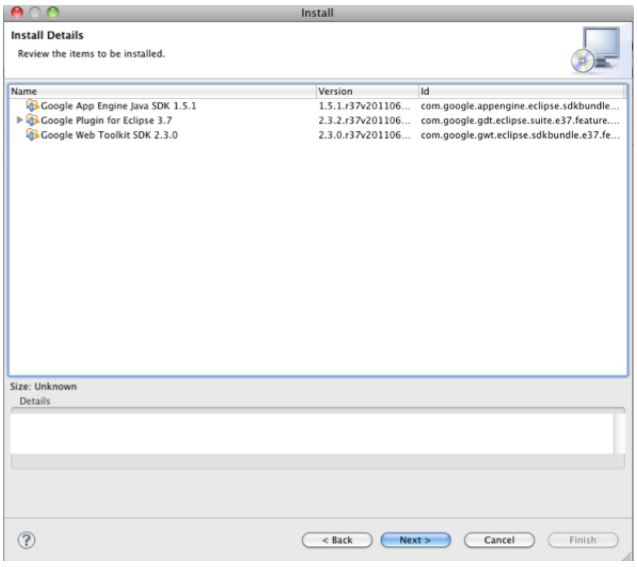

Figura 2.9: Elements a instal·lar

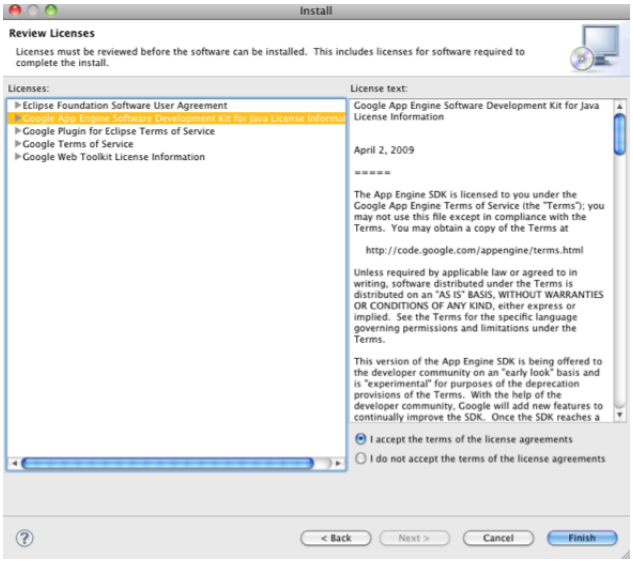

Figura 2.10: Acords de llicència

Figura 2.11: Reiniciar Eclipse

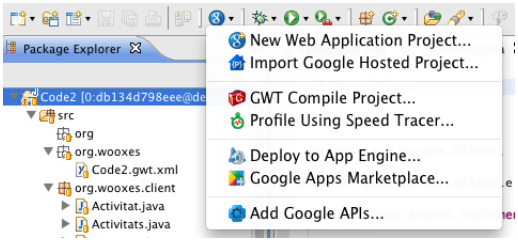

Figura 2.12: Icona Google Web Toolkit

2.4.2 Classes actualitzades [5]

En aquest apartat es mostra les classes més rellevants que s'han actualitzat de la versió 1.4 a la versió 2.4 amb les seves característiques:

- **ClickHandler** és la classe actualitzada de ClickListener. És un controlador pels events ClickEvent on utilitza el mètode onClick(ClickEvent event) que es dispara quan es fa clic en un event.

  Exemple:

GWT 1.4

```
Button enviar = new Button("Enviar");
enviar.addClickListener(new ClickListener(){
  public void onClick(Widget sender){
    if(area.getText().equals("")){
      Window.alert("No es pot enviar un comentari
          buit");
    }
    else if (llista_tipus.getSelectedIndex() == 0){
      Window.alert("Selecciona tipus de comentari");
    }
    else{
      h_accio.setValue("afegir");
      h_comentari.setValue(area.getHTML());
      form.submit();
    }
  }
});
```

GWT 2.4

```
Button enviar = new Button("Enviar");
enviar.addClickHandler(new ClickHandler(){
  public void onClick(ClickEvent sender){
    if(area.getText().equals("")){
      Window.alert("No es pot enviar un comentari
          buit");
    }
    else if (llista_tipus.getSelectedIndex() == 0){
      Window.alert("Selecciona tipus de comentari");
    }
    else{
      h_accio.setValue("afegir");
      h_comentari.setValue(area.getHTML());
      form.submit();
    }
  }
});
```

- En la nova versió de GWT, la funcionalitat de FormHandler es divideix en dues interfícies (FormPanel.SubmitHandler i FormPanel.SubmitCompleteHandler) definides com a classes estàtiques internes i públiques de la classe **FormPanel**.

Exemple:

GWT 1.4

```
fpCrearAssignatura.addFormHandler(new FormHandler(){
  public void onSubmit(FormSubmitEvent event){
    if(boxCrearAssignatura.getText().length() == 0 ){
      Window.alert("Nom buit");
      event.setCancelled(true);
    }
  }
  public void onSubmitComplete(
      FormSubmitCompleteEvent event){
      String resposta = event.getResults();
      if(!resposta.startsWith("ok")){
        Window.alert(resposta);
      }
      getActivitatsService().getAssignatures(
          mostrarAssignatures());
  }
});
```

GWT 2.4

```
fpCrearAssignatura.addSubmitHandler(new FormPanel.
    SubmitHandler(){
  public void onSubmit(SubmitEvent event){
    if(boxCrearAssignatura.getText().length() == 0 ){
      Window.alert("Nom buit");
      event.isCanceled();
    }
  }
});

fpCrearAssignatura.addSubmitCompleteHandler(new
    FormPanel.SubmitCompleteHandler(){
  public void onSubmitComplete(SubmitCompleteEvent
      event){
      String resposta = event.getResults();
      if(!resposta.startsWith("ok")){
        Window.alert(resposta);
      }
      getActivitatsService().getAssignatures(
          mostrarAssignatures());
  }
});
```

- **OpenHandler** és la interfície actualitzada de DisclosureHandler. És una interfície controladora per events OpenEvents que representen els events oberts. Els mètodes que s'utilitzaven eren onClose(DisclosureEvent event) que s'activava quan el panel estava tancat i onOpen(DisclosureEvent event) quan el panel està obert, ara s'ha actualitzat per onOpen(OpenEvent<T> event).

  Exemple:

GWT 1.4

```
dp.addEventHandler(new DisclosureHandler(){
  public void onClose(DisclosureEvent event){
  }
  public void onOpen(DisclosureEvent event){
    box.setText(b);
});
```

GWT 2.4

```
dp.addOpenHandler(new OpenHandler<DisclosurePanel>(){
  @Override
  public void onOpen(OpenEvent<DisclosurePanel> event
      ){
    box.setText(b);
});
```

- **ValueChangeHandler** és la interfície actualitzada de HistoryListener. Aquesta interfície s'utilitzava per rebre notificacions de canvis en l'estat de l'historial del navegador. El mètode que utilitzava era el onHistoryChanged(java.lang.String historyToken) que s'activava quan l'usuari feia un clic als botons d'endavant i d'endarrere del navegador.

- **LoadHandler** és la interfície actualitzada de LoadListener. Aquesta interfície és una controladora d'events LoadEvents. Els mètodes que s'utilitzaven eren onError(Widget sender) i onLoad(Widget sender i s'han actualitzat per onLoad(LoadEvent event).

  Exemple:

GWT 1.4

```
imatge.addLoadListener(new LoadListener(){
  public void onError(Widget sender){
  //Sinó es carrega l'imatge,es visualitzarà el text
    textAlternatiu.setVisible(true);
  }
  public void onLoad(Widget sender){
  }
});
```

GWT 2.4

```
imatge.addLoadHandler(new LoadHandler(){
  public void onError(LoadEvent sender){
  //Sinó es carrega l'imatge,es visualitzarà el text
    textAlternatiu.setVisible(true);
  }
  public void onLoad(LoadEvent sender){
  }
});
```

- **ClientBundle** és la interfície actualitzada de ImageBundle. La utilitat d'aquesta interfície és similar a la de ImageBundle. El *Source* especifica la ruta d'ubicació dels recursos. Durant l'execució, les funcions retornaran un objecte que pot ser utilitzat per accedir a les dades.

  Exemple:

GWT 1.4

```
public interface Images extends ImageBundler{
  /**
  * @gwt.resource org/wooxes/client/icons/bold.gif
  */
  AbstractImagePrototype bold();
```

GWT 2.4

```
public interface Resources extends ClientBundle{

  @Source("org/wooxes/shared/icons/bold.gif");
  ImageResource bold();
```

- **ChangeHandler** és la interfície actualitzada de ChangeListener que utilitza el mètode onChange(Widget sender). Aquesta nova interfície es una controladora pels events ChangeEvent i utilitza el mètode onChange(ChangeEvent event) que es crida quan un event de canvi es llença.

  Exemple:

GWT 1.4

```
private class EventListener implements ClickListener,
     ChangeListener, KeyboardListener {
  public void onClick(Widget sender){
    if(sender == bold){
      basic.toggleBold();
    } else if (sender == italic){
      basic.toggleItalic();
```

GWT 2.4

```
private class EventHandler implements ClickHandler,
    ChangeHandler, KeyUpHandler {
  public void onClick(ClickEvent sender){
    if(sender == bold){
      basic.toggleBold();
    } else if (sender == italic){
      basic.toggleItalic();
```

2.5 Adaptació de l'aplicació a la plataforma Google App Engine

Google App Engine és un servei de *Cloud Computing* gratuït per aplicacions web programades amb Java, Python o el llenguatge de Google anomenat "Go". Proporciona als desenvolupadors una infrestructura per instal·lar i executar aplicacions web en un "núvol".

En el cas d'una aplicació web desenvolupada amb Java, és possible desplegar-la sobre la infraestructura que Google proporciona i accedir a ella mitjantçant un domini gratuït del tipus *http://name.appspot.com*

Per començar a utilitzar App Engine, el primer pas es registrar-se a *http://appengine.google.com* on és necessari tenir un compte Google. A continuació es crea l'aplicació i es verifica el compte mitjançant l'enviament per part de Google d'un codi per SMS al mòbil que l'usuari facilita sense cap cost. Aquest procés de verificació solament s'ha de fer al crear la primera aplicació. Si la verificació es correcta, el següent pas es introduir un identificador únic que s'afegirà al fitxer appengine.xml i a la configuració de l'aplicació, Application Identifier, i el títol de l'aplicació que s'ha de crear, tal i com es mostra a la figura 2.13.

Un cop registrada l'aplicació, la Consola d'Administració de Google App Engine (Figura 2.16), ofereix un accés total als paràmetres de configuració de l'aplicació i permet realitzar un seguiment de les execucions o peticions realitzades durant les últimes 24 hores. Per poder accedir-hi és necessari autenticar-se des del portal que ofereix Google i un cop registrat, seleccionar una de les aplicacions desenvolupades.

En concret, des de la Consola d'Administració és possible executar les següents funcionalitats:

- Configurar els paràmetres de l'aplicació.
- Gestionar els permissos de l'aplicació per permetre l'accés a diferents usuaris assignant-los-hi un rol: Administrador, Desenvolupador o Observador.
- Portar un control de versions del codi.
- Provar noves versions de l'aplicació i canviar la versió disponible pels usuaris en cada moment.
- Navegar per l'emmagatzemament de dades de l'aplicació i administrar els índexs.
- Comprovar l'estat de les tasques programades de l'aplicació.

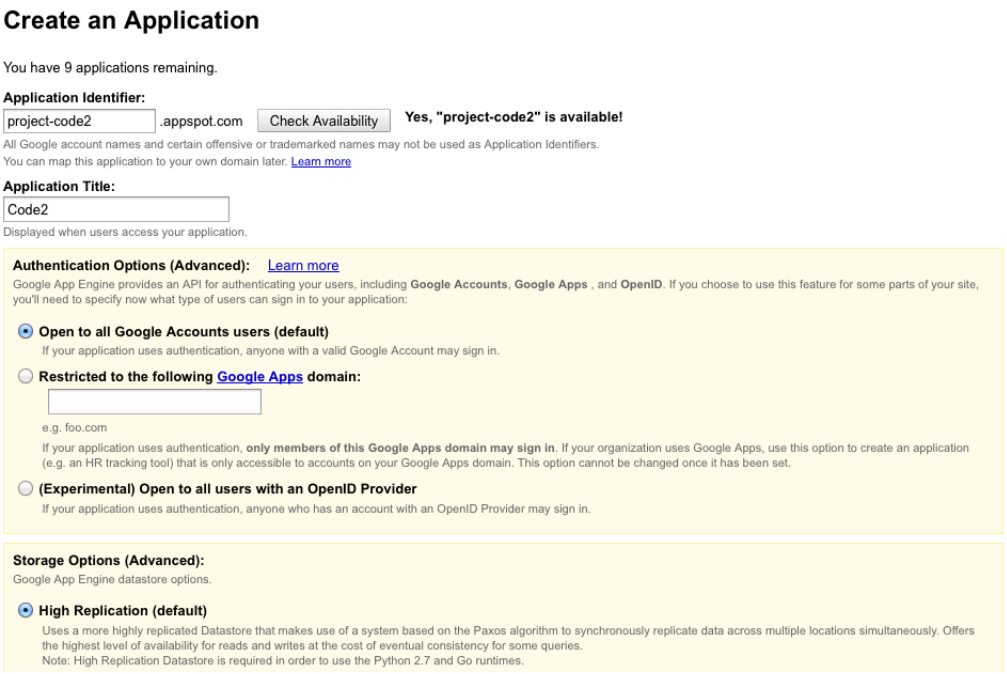

Figura 2.13: Creació de l'aplicació

- Visualitzar els registres d'errors i les dades d'accés de les peticions realitzades, per analitzar el tràfic durant un període de temps determinat.
- Consultar la gràfica i les estadístiques referents als recursos (emmagatzemament, temps de CPU, número de peticions, etc) consumits i disponibles per l'aplicació.

Les primeres 10 aplicacions són gratuïtes, sempre i quan no es superi el consum de memòria, CPU, quantitat de visites, entre d'altres. Per les aplicacions de prova no hi ha cap problema, ja que el consum de recursos es queda molt per sota dels màxims. A més a més, el contador es reseteja cada 24 hores.

El següent pas és activar l'App Engine a l'aplicació en l'entorn d'Eclipse. Des de les propietats de l'aplicació, s'ha d'activar el SDK (appengine-java-sdk-1.7.0) introduint el Application ID (identificador que hem creat al registrar l'aplicació: project-code2) i el número de la versió tal i com es pot veure a la figura 2.17.

Un cop tot configurat, ja es pot realitzar i aplicar tots els canvis necessaris per tal d'adaptar Google App Engine a l'aplicació.

En les següents seccions, es detallen els canvis més significatius que hi ha a l'aplicació per tal d'adaptar Google App Engine.

1. Modificació del sistema de base de dades: Adaptació de Java Data Objects (JDO).
2. Modificació de l'emmagatzemament de fitxers: Servei d'emmagatzemament de blobs.

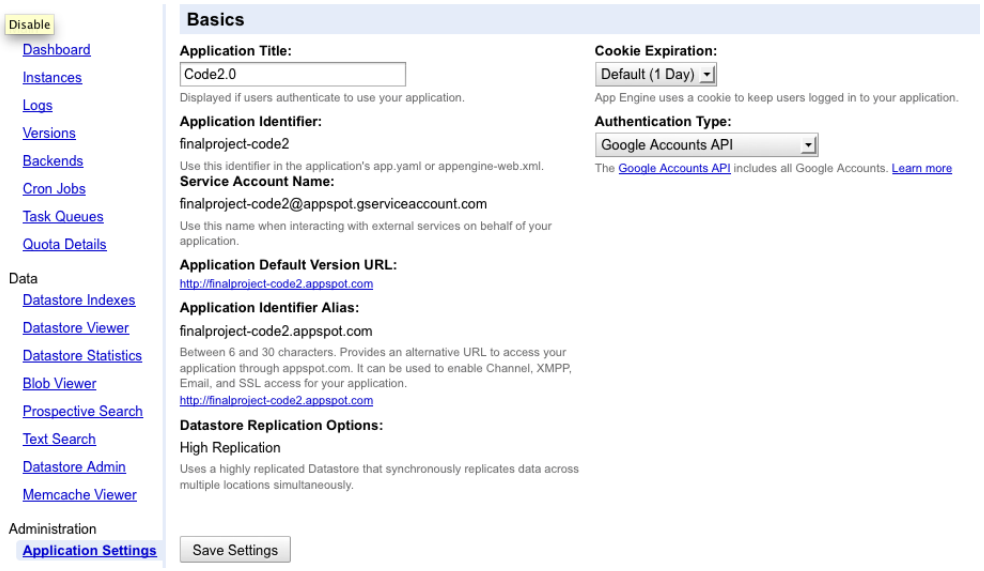

Figura 2.14: Paràmetres de l'Aplicació

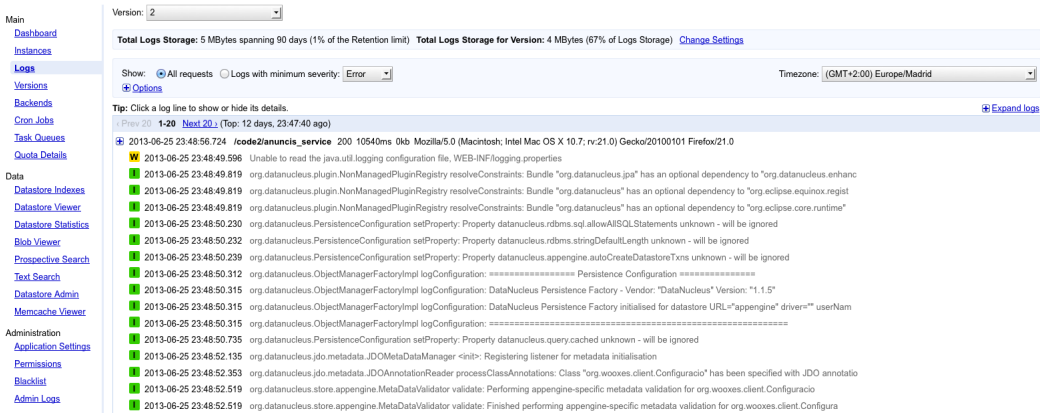

Figura 2.15: Logs

2.5.1 Instal·lació del plugin GAE per Eclipse

El "plugin" de GAE per Eclipse permet desenvolupar aplicacions d'App Engine, de forma àgil i eficaç. Incorpora uns botons especials a la barra d'eines, per facilitar la creació, compilació i desplegament de les aplicacions.

El complement està desenvolupat per Google i és objecte de freqüents millores i actualitzacions. La versió utilitzada en la codificació del projecte ha estat la Google Plugin for Eclipse 4.2.

2.5.2 Modificacions a l'aplicació

2.5.2.1 Modificació del sistema de base de dades de l'aplicació

L'emmagatzemament de dades que utilitza GAE està dissenyat per optimitzar la lectura i l'execució de consultes, guardar objectes de dades (coneguts amb el nom d'entitats), i executar

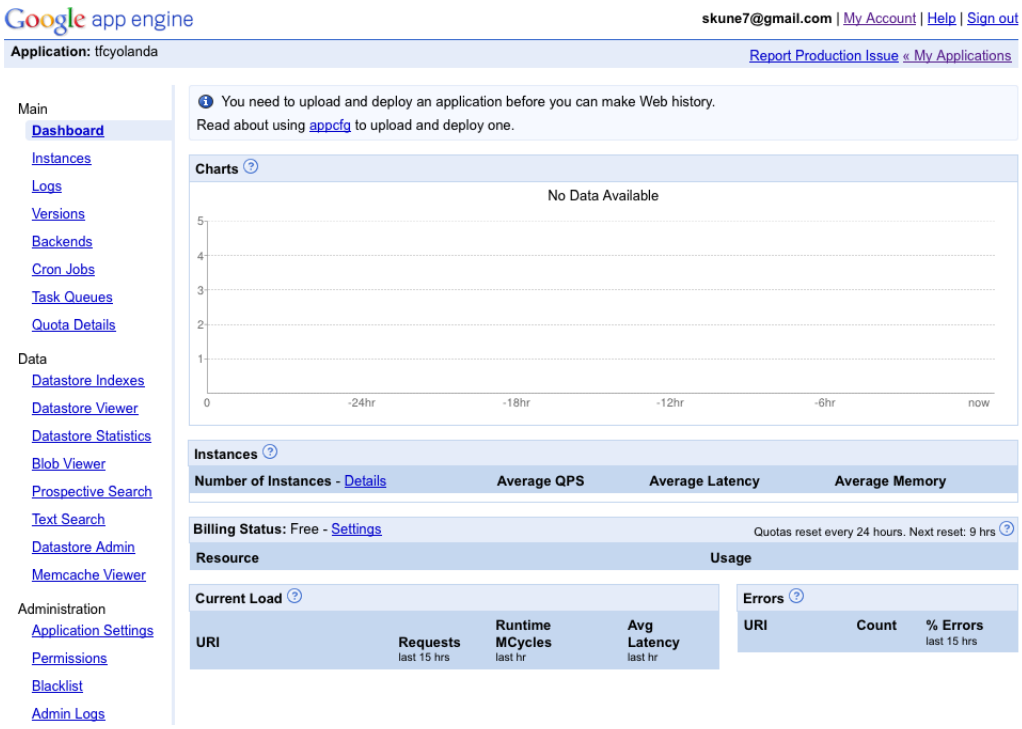

Figura 2.16: Panel de control App Engine

diferents operacions en una única transacció, per poder-les recuperar en cas d'error. Això resulta especialment útil en el cas d'aplicacions web distribuïdes, on és possible que varis usuaris accedeixin als mateixos objectes de dades de forma concurrent.

A diferència de les bases de dades tradicionals, l'emmagatzemament de dades utilitza una arquitectura distribuïda per gestionar conjunts de dades de gran tamany. Una aplicació App Engine pot optimitzar la forma en que es distribueixen les dades mitjançant la descripció de relacions entre objectes de dades i la definició d'índexs per consultes.

L'emmagatzemament de dades App Engine és totalment consistent, però això no implica que es tracti d'una base de dades relacional. Encara que la seva interfície utilitzi funcions de les bases de dades tradicionals, les característiques úniques de l'emmagatzemament de dades suposen una forma diferent de dissenyar i gestionar les dades. Per exemple, les entitats de l'emmagatzemament de dades no tenen un esquema únic: dos entitats del mateix tipus no estan obligades a contindre les mateixes propietats o utilitzar els mateixos tipus de valors. En conseqüència, la pròpia aplicació es la única responsable de garantitzar que les entitats cumpleixin un determinat esquema quan es necessari.

Google App Engine, a través de l'entorn de desenvolupament SDK Java, inclou implementacions dels *Objectes de Dades Java (JDO)* i de les interfícies de l'*Api de persistencia de Java (JPA)* per crear i persistir dades.

Java Data Objects (JDO) [7]

Pel desenvolupament del projecte s'ha optat per utilitzar *Java Data Objects* per la persistència de dades. JDO ofereix avantatges respecte a JPA, tal i com es pot comprovar a la següent taula

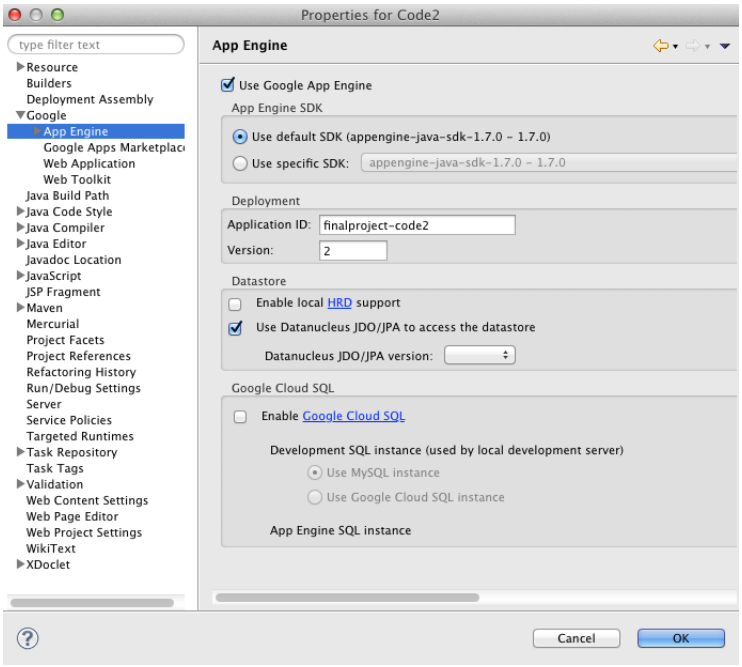

Figura 2.17: Configuració App Engine

Figura 2.18: Botons plugin GAE

comparativa. Disposa d'una gran quantitat de documentació per la utilització dins de GAE junt amb un elevat grau de desenvolupament en GAE. Aquestes característiques han estat les raons fonamentals per les que finalment s'ha seleccionat JDO enlloc de JPA.

Comparativa entre Java Data Objects (JDO) i Java Persistence API (JPA):

	Java Data Objects	**Java Persistence API**
Requisits SDK Java	1.3 o superior	1.5 o superior
RDBM (Relational Data Base Modeling	Suportat	Suportat
ORM (Object Role Modeling)	Suportat	No suportat
Catàleg de tipus de dades suportades	Gran	Mitjà
DataNucleus	Suportat	No suportat
Grau de desenvolupament en GAE	Alt	Mitjà-Baix

JDO es una especificació d'emmagatzemament d'objectes en Java, la qual no es limita solament a l'emmagatzemament de dades de base de dades relacionals, podent utilitzar-se també contra fitxers XML, documents OpenOffice, objectes JSON, etc. Aquesta característica facilita la integració entre JDO i el datastore. No obstant, si que utilitza un API de nivell inferior, tal i com es pot observar a la figura 2.19.

Els canvis més significatius que s'han realitzat per l'emmagatzemament de dades, es poden classificar en 5 apartats:

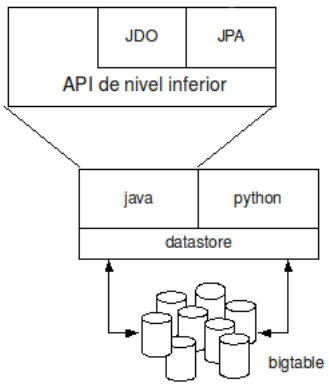

Figura 2.19: Esquema JDO

- Classe de dades
- PersistenceManager
- Creació, obtenció i eliminació de dades
- Relacions d'entitats
- Transaccions

A continuació s'explica cadascun dels punts nombrats anteriorment:

- **Classes de dades**

JDO s'utilitza per emmagatzemar objectes de dades simples. Cada objecte que guarda JDO es converteix en una entitat del datastore d'App Engine. El tipus de l'entitat s'obté a partir del nom simple de la classe. Cada camp persistent de la classe representa una propietat de l'entitat, amb un nom de propietat idèntica al nom del camp.

Per declarar que una classe pot ser recuperada i emmagatzemada mitjançant el suport de JDO, s'assigna una anotació a la classe: **@PersistenceCapable**.

Els camps de la classe de dades que es desitgen guardar al datastore, s'han de declarar com a camps persistents. Per fer-ho, s'utilitza l'anotació: **@Persistent**.

Una classe de dades ha de tenir un camp dedicat a l'emmagatzemament de la clau principal de l'entitat corresponent al datastore. El camp de la clau principal s'identifica mitjançant l'anotació: **@PrimaryKey**. Amb Google App Engine, hi ha 4 maneres de definir claus primàries:

- Com objectes de tipus Long, on l'id es genera de manera automàtica.

- Com objectes de tipus String, on l'id l'ha de generar el propi usuari.

- Com objectes de tipus Key numèric, on l'id es genera de manera automàtica.

- Com objectes de tipo Key, representats com una cadena, on l'id es genera de manera automàtica.

A continuació podem veure un exemple dels diferents tipus d'anotacions:

Anunci.java

```
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Anunci implements Serializable {

private static final long serialVersionUID = 1L;

 @PrimaryKey
 @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
 @Extension(vendorName = "datanucleus", key = "gae.encoded-pk", value = "true")
 private String id;

 @Persistent
 private String titol;

 @Persistent
 private String text;

 @Persistent
 private Date data;
```

- **Persistence Manager**

Una aplicació interactua amb JDO utilitzant una instància de la classe PersistenceManager. Aquesta instància s'obté a l'invocar un mètode en una instància de la classe PersistenceManagerFactory. Aquesta utilitza la configuració de JDO per crear instàncies de PersistenceManager.

Donat que la instància de PersistenceManagerFactory tarda un temps en iniciar-se, les aplicacions han de reutilitzar sempre la mateixa instància. Per garantitzar que això succeeixi, es genera una excepció quan l'aplicació crea més d'una instància de PersistenceManagerFactory (amb el mateix nom de configuració). Una forma senzilla de gestionar la instància de PersistenceManagerFactory es crear una classe única amb una instància estàtica.

PMF.java

```
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;
public final class PMF {
 private static final PersistenceManagerFactory pmfInstance =
     JDOHelper.getPersistenceManagerFactory("transactions-
     optional");

  private PMF() {}
  public static PersistenceManagerFactory get() {
      return pmfInstance;
  }
}
```

AnunciServiceImpl.java

```
public List<Anunci> getAnuncis()
PersistenceManager pm = PMF.get().getPersistenceManager();
ArrayList<Anunci> v = null;

try {
Query query = pm.newQuery(Anunci.class);
query.setOrdering("data DESC");
v = new ArrayList<Anunci>((List<Anunci>) query.execute());
}
catch(Exception e) {
  e.printStackTrace();
}
finally{
if(pm.currentTransaction().isActive())
  pm.currentTransaction().rollback();
if (!pm.isClosed())
  pm.close();
}
return v;
}
```

En resum,

PersistenceManager és la interfície primària de JDO i proporciona mètodes per crear les consultes, transaccions i gestiona el cicle de vida de les instàncies persistents.

PersistenceManagerFactory és el responsable de configurar i crear les instàncies PersistenceManager. Representa la implementació particular de JDO utilitzada. Disposa de mètodes per determinar les propietats i característiques de la implementació, dels mètodes per definir la connexió amb l'emmagatzemament i determina la configuració de l'entorn sobre el que s'executa les instàncies de la classe PersistenceManager.

- **Creació, obtenció i eliminació de dades**

Per guardar un objecte persistent al datastore, s'ha d'invocar el mètode **makePersistent()** de PersistenceManager i transmetre la instància. Aquest mètode retorna els resultats quan l'objecte s'ha guardat i els índexs s'han actualitzat correctament.

```
public class CrearAnunciServlet extends HttpServlet {

 PersistenceManager pm = PMF.get().getPersistenceManager();
 Transaction tx = pm.currentTransaction();

 try {
   tx.begin();
   Anunci auxanunci = new Anunci();
   auxanunci.setTitol(BackSlashes.add(request.getParameter("titol
       ")));
   auxanunci.setText(BackSlashes.add(request.getParameter("
       descripcio")));
   auxanunci.setData(new Date());

   if(auxanunci.getTitol().length() != 0){
     pm.makePersistent(auxanunci);
   out.println("S'ha creat l'anunci correctament");
   tx.commit();
   } else out.println("Introdueixi les dades de l'anunci");
} catch (Exception e) {
    e.printStackTrace();
    out.println("Error: No s'ha pogut crear l'anunci");
} finally {
if(tx.isActive())
    pm.currentTransaction().rollback();
if(!pm.isClosed())
    pm.close();
}
}
}
```

Per recuperar un objecte a partir de la seva clau, s'utilitza el mètode **getObjectById()** de PersistenceManager. El mètode obté la classe de l'objecte i la clau. App Engine ha de ser capaç d'obtenir la clau complerta a partir del nom de la classe i del valor proporcionat.

```
public class EliminarAssignaturaServlet extends HttpServlet {

 PersistenceManager pm = PMF.get().getPersistenceManager();
 Transaction tx = pm.currentTransaction();

 try {
   tx.begin();
   Assignatura assig = pm.getObjectById(Assignatura.class, new
        String(request.getParameter("id")));

   if(assig != null)
    pm.deletePersistent(assig);
    out.println("L'assignatura s'ha esborrat correctament");
    tx.commit();
} catch (Exception e) {
    e.printStackTrace();
    out.println("Error: No s'ha pogut eliminar l'assignatura");
} finally {
if(tx.isActive())
    pm.currentTransaction().rollback();
if(!pm.isClosed())
    pm.close();
}
}
}
```

Un sistema d'actualitzar un objecte amb JDO es extreure l'objecte, i a continuació, modifica'l mentre la interfície PersistenceManager que ha retornat l'objecte segueix oberta. Els canvis persisteixen una vegada que PersistenceManager es tanca.

```
public class ModificarAnunciServlet extends HttpServlet {

 PersistenceManager pm = PMF.get().getPersistenceManager();
 Transaction tx = pm.currentTransaction();

 try {
   tx.begin();
   Anunci auxanunci = new Anunci();
   auxanunci.setId(request.getParameter("id"));
   auxanunci = pm.getObjectById(Anunci.class,auxanunci.getId());
   auxanunci.setTitol(request.getParameter("titol"))
   auxanunci.setText(request.getParameter("descripcio"));

   pm.makePersistent(auxanunci);

   out.println("L'anunci s'ha modificat correctament");
   tx.commit();
} catch (Exception e) {
    e.printStackTrace();
    out.println("Error: No s'ha pogut modificar el titol de l'
        anunci");
} finally {
if(tx.isActive())
    pm.currentTransaction().rollback();
if(!pm.isClosed())
    pm.close();
}
}
}
```

En el cas d'eliminar un objecte del magatzem de dades, s'invoca el mètode **deletePersistent()** de PersistenceManager amb l'objecte:

```
Assignatura assig = pm.getObjectById(Assignatura.class, new
    String(request.getParameter("id")));

if(assig != null)
 pm.deletePersistent(assig);
 out.println("L'assignatura s'ha esborrat correctament");
```

- **Relació d'entitats**

La implementació de la interfície de JDO amb App Engine, pot modelar relacions entre objectes persistents mitjançant camps de tipus d'objecte. Les relacions poden ser de tipus 1:N o 1:1 tant unidireccional com bidireccional, però no es permeten les relacions N:M. App Engine crea entitats relacionades amb grups d'entitats de forma automàtica per fer possible l'actualització de varis objectes relacionats, encara que l'aplicació s'encarrega de decidir quan s'ha d'utilitzar les transaccions del datastore.

1. Relacions 1:1

   Es crea una relació de propietat d'un a un entre dos objectes persistents mitjançant un camp on el tipus es la classe de la classe relacionada. Els objectes persistents es representen com dos entitats diferents al datastore.

   Per crear una relació bidireccional un a un, el camp de la classe secundària ha de tenir una annotació **@Persistent** amb l'argument **mappedBy = "..."**, on el valor es el nom del camp a la classe principal.

   Empleat.java

```
import InfoContacte;

// ...
    @Persistent(defaultFetchGroup = "true")
    private InfoContacte infocontacte;
```

   InfoContacte.java

```
import Empleat;

// ...
   @Persistent(mappedBy = "infocontacte")
   private Empleat empleat;
```

2. Relacions 1:N

   Per crear una relació d'un a varis entre objectes d'una classe i varis objectes d'una altra, s'utilitza una col·lecció de la classe relacionada. A la classe principal s'utilitza l'annotació **@Persistent (mappedBy = "...")** en el que el valor es el nom del camp de la classe secundària.

   Assignatura.java

```
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Assignatura implements Serializable {
  private static final long serialVersionUID = 1L;

  @Persistent
    private String nom;

  @PrimaryKey
  @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
  @Extension(vendorName = "datanucleus", key = "gae.encoded-pk",
      value = "true")
    private String id;

  @Persistent(mappedBy="assignatura")
  @Order(extensions = @Extension(vendorName="datanucleus", key="
      list-ordering", value="nom asc"))
  private List<Activitat> activitats;
```

   Activitat.java

```
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class Activitat implements Serializable {
  private static final long serialVersionUID = 1L;

  @PrimaryKey
  @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
  @Extension(vendorName = "datanucleus", key = "gae.encoded-pk",
      value = "true")
    private String id;

  @Persistent
    private String blobid;

  @Persistent
    private String nom;

  @Persistent
    private String descripcio;

  @Persistent
    private Date data;

  @Persistent
    private String adjunt;

  @Persistent
    private Assignatura assignatura;
```

En aquest exemple s'indica que una Assignatura pot tenir vàries Activitats, mentre que una Activitat pertany a una Assignatura. A la classe Assignatura també s'indica l'ordenació, amb l'annotació **@Order**, que tindrà la llista d'activitats quan es guardi l'objecte principal al datastore. JDO requereix que les bases de dades conservin un ordre, emmagatzemant la posició de cada objecte com una propietat.

- **Transaccions**

El datastore d'App Engine admet transaccions. Una transacció es una operació o un conjunt d'operacions on es poden realitzar diferents càlculs dins d'una mateixa transacció. Si la transacció es resolt correctament, tots els efectes desitjats s'aplicaran al datastore, en canvi si no es així, no s'aplicaran. Per exemple, una operació de creació o modificació pot realitzar-se correctament o no realitzar-se en absolut.

Les transaccions són funcions opcionals del datastore, es a dir, no es necessari utilitzar-les per realitzar operacions, ja que el datastore es transaccional, això significa que evita que es realitzin inconsistències entre les dades.

Utilitzant l'API de JDO es pot declarar de manera explícita una transacció tal i com es mostra a continuació:

```
public class EliminarAssignaturaServlet extends HttpServlet {

 PersistenceManager pm = PMF.get().getPersistenceManager();
 Transaction tx = pm.currentTransaction();

 try {
   tx.begin();

   //codi

   tx.commit();

} catch (Exception e) {
    e.printStackTrace();

} finally {
if(tx.isActive())
   pm.currentTransaction().rollback();
if(!pm.isClosed())
   pm.close();
}
}
}
```

On:

- transaction.begin() inicia la transacció.
- transaction.commit() aplica els canvis efectuats des de que s'ha iniciat la transacció i ha finalitzat.
- transaction.rollback() no guarda els canvis efectyrats des de que ha iniciat la transacció i ha finalitzat.

2.5.2.2 Modificació de l'emmagatzemament de fitxers

Tal i com s'ha comentat anteriorment, Google App Engine no suporta l'emmagatzemament de fitxers d'una forma estàndard. Per suplir aquesta carència, aquesta plataforma proporciona una sèrie de mecanismes i serveis alternatius que es descriuen a continuació:

- **Bigtable**

  És la forma més senzilla d'emmagatzemar fitxers, ja que simplement s'ha de guardar el fitxer en una propietat de tipus byte[]. El principal inconvenient d'aquest mètode és que la mida màxima del fitxer no pot ser superior a 1 MB.

- **Blobstore**

  Aquest servei permet a les aplicacions guardar objectes de fins a 2 gigabytes a través de l'enviament d'un formulari web i/o una sol·licitud HTTP POST.

  Les avantatges d'aquest servei són:

  – No s'utilitza la quota del servei de DataStore, sinó que es contabilitza en un servei diferent.
  – No suposa un cost la utilització de la CPU per recuperar els fitxers.
  – Els fitxers es recuperen de manera més ràpida ja que es descarreguen del servidor més pròxim geogràficament.

  No obstant, té una sèrie d'inconvenients:

  – No inclou un sistema de seguretat per accedir als fitxers. Això involucra que, per exemple, manipulant els paràmetres de la URL es poden obtenir el fitxers de l'usuari.
  – El mòdul de facturació ha d'estar habilitat.
  – Quan s'elimina un blob, no s'elimina immediatament el fitxer, aquest estarà disponible durant un temps.

Utilizant el servei Blobstore

Ja que la mida dels fitxers que poden carregar els usuaris de l'aplicació pot superar 1 MB, s'ha optat per implementar l'emmagatzemament de fitxers a través del servei de BlobStore.

Aquest servei s'invoca desde el camp "Input File" dels formularis HMTL a través d'una petició POST dirigida a una adreça concreta. Un cop s'han carregat aquests fitxers, el servei se'n ocupa de gestionar-los per a que el desenvolupador pugui tractar-los adequadament segons les necessitats de la seva aplicació. Per gestionar aquests fitxers (o Blobs), s'utilitza la classe BlobInfo que conté informació sobre el nom, mida, etc.

El procediment detallat que utilitza l'aplicació s'explica a continuació:

- Es defineix l'adreça de l'acció del formulari HTML a partir de la cadena obtinguda de cridar al mètode **blobstoreService.createuploadUrl**.

- Quan l'usuari envia el formulari, el servei de BlobStore processa la sol·licitud POST, que crea un mapa amb tants Blobs com fitxers s'han enviat des del formulari. Aquest servei també crea un registre d'informació per cada blob. D'aquesta forma s'abstrau al desenvolupador de tota la implementació de baix nivell.
  S'ha de tenir en compte que aquest mètode guarda automàticament tots els fitxers enviats des del formulari. Si no es vol conservar algun d'aquests fitxers (o Blobs), s'ha d'eliminar manualment per evitar ocupar espai innecessari.

- Un cop processada la informació del formulari, el mètode **getUploads** retorna uns conjunts de blobs que s'han emmagatzemat. L'objecte **Map** és una llista que associa els noms del camps amb els blobs que conté.

La classe encarregada de gestionar la descàrrega dels fitxers que s'han emmagatzemat com a Blobs és la DescargarArxiu. Aquesta classe proporciona els fitxers a través de peticions GET on el paràmetre "id" indica l'identificador del fitxer que es vol obtenir.

Utilitzant la Base de Dades a través d'una capa d'abstracció

Tot i els avantatges que comporta la utilització del servei Blobstore respecte la base de dades al moment de guardar fitxers, aquest presenta una limitació que feia impossible mantenir l'esquema de funcionament del mòdul de comentaris de l'aplicació sense realitzar grans canvis en la seva implementació.

L'aplicació guarda els comentaris que realitzen els usuaris sobre un fitxer determinat creant un document en format XML amb el mateix nom que el fitxer al que es refereix i la extensió ".comment". Un cop creat, es modificarà a mesura que els usuaris creïn nous comentaris. Aquest esquema de funcionament és incompatible amb el servei Blobstore, ja que aquest no permet modificar un fitxer que ja s'ha guardat algun cop en el seu sistema.

Una possible solució seria crear una còpia temporal del document original, afegir-hi el comentari i sobreescriure'l per l'original, però comportaria una sèrie d'inconvenients, com per exemple la necessitat d'implementar un sistema de sincronització per tal d'evitar condicions de carrera en el cas en que múltiples usuaris afegeixin un comentari al mateix temps. A més, tant el rendiment de l'aplicació com el cost monetari es podrien veure afectats per la sobrecàrrega que suposaria el fet d'haver de crear una còpia del document per cada comentari nou que s'afegeix.

Així doncs, l'única solució que es planteja es guardar aquests documents sobre la base de dades de la plataforma App Engine. Tot i això, aquest enfoc suposaria una modificació important de la implementació del mòdul de comentaris, ja que aquest està basat en l'existència d'un sistema de fitxers. Per tal d'evitar tots aquests canvis, s'ha creat una capa d'abstracció amb la finalitat de poder interactuar amb la base de dades com si es tractés d'un sistema de fitxers. D'aquesta forma, s'ha pogut mantenir pràcticament tota la implementació del mòdul de comentaris, sense afectar al rendiment de l'aplicació.

Aquesta capa d'abstracció està formada bàsicament per dues classes: "DBFile" i "DBFileSystemImpl".

La classe "DBFile" representa un fitxer associat a la base de dades. Bàsicament és un JavaBean amb els següents mètodes:

- **String getFileID() / void setFileID(String fileID):** Aquests dos mètodes serveixen per obtenir i establir l'identificador del fitxer. Aquest identificador es podria considerar com "el nom del fitxer" d'un sistema de fitxers tradicional.
- **String getFileContent() / void setFileContent(String fileContent):** Aquests dos mètodes serveixen per obtenir i establir el contingut del fitxer.

Per altra banda, la classe "DBFileSystemImpl" representa el sistema de fitxers associat a la base de dades. Aquesta classe implementa els següents mètodes:

- **DBFile newFile(String fileID):** Aquest mètode permet crear un nou fitxer dins el sistema de fitxers.
- **DBFile openFile(String fileID):** Aquest mètode permet obre un fitxer del "DBFileSystemImpl", retornant una instància de la classe "DBFile".
- **void saveFile(DBFile file):** Aquest mètode guarda un objecte "DBFile" en el sistema de fitxers.
- **boolean exists(String fileID):** Aquest mètode comprova si el fitxer amb l'identificador "fileID" existeix en el sistema de fitxers.

2.5.2.3 Autenticació d'usuaris

Google App Engine ofereix diferents serveis basats en la infraestructura de Google en el que les aplicacions poden accedir mitjantçant una sèrie de biblioteques incloses al SDK. Un d'ells és el servei d'usuaris que permet integrar l'aplicació al compte d'usuari de Google. Amb aquest servei, els usuaris poden utilitzar els seus comptes de Google existents per accedir a l'aplicació.

Google ofereix diferents sistemes per l'autenticació d'usuaris en una aplicació. Solament es pot utilitzar una forma d'autenticació per cada aplicació.

- **Google Acounts**

  És el sistema d'accés unificat de Google. L'únic que necessita un usuari és una direcció de correu electrònic vàlida per obtenir un compte de Google.

- **Comptes dels pròpis dominis de Google Apps**

  Els usuaris de Google Apps poden optar per restringir tota la seva aplicació web o part d'ella sols a aquelles persones que disposin d'una direcció de correu electrònic vàlida en el seu domini.

- **Identificadors OpenID**

  OpenID és una tecnologia oberta que s'utilitza per l'autenticació d'usuaris en diferents serveis web. Quan un usuari crea un compte en un servei que actua com a proveïdor de OpenID, pot utilitzar una URL única d'accés a aquest servei com identificador OpenID per accedir a qualsevol altre servei.

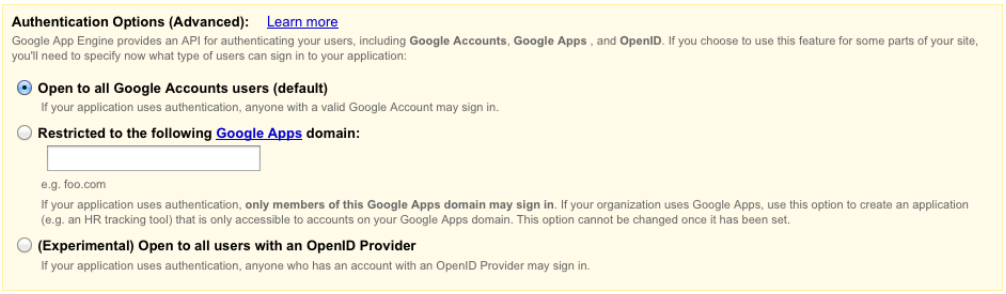

Figura 2.20: Opcions d'autenticació d'usuaris

El sistema d'autenticació s'elegeix en el moment de crear una aplicació a Google App Engine. De forma predeterminada, l'aplicació utilitzarà Google Accounts per l'autenticació tal i com es mostra en la figura 2.20:

S'ha de tenir en compte que una vegada creada l'aplicació, les opcions de canviar l'autenticació són limitades. Concretament, solament es pot canviar entre Google Accounts i OpenID.

Una aplicació pot detectar si l'usuari actual ha iniciat una sessió i pot redirigir-lo a la pàgina d'accés corresponent per a que accedeixi al seu compte o, si l'aplicació utilitza l'autentificació de Google Accounts, per a que creï un compte nou.

La interfície encarregada de proporcionar la informació d'usuari, així com autentificar-se a Google, es *UserService*. *UserService* permet:

- Per una banda, verificar si l'usuari es troba loguejat.
- Per altra banda, generar URLs de login i logout per poder autenticar-se a Google i un cop registrat, redirigir a la URL que es desitja.
- També ens permet poder determinar si l'usuari és o no administrador de l'aplicació, el que facilitarà la implementació d'àrees exclusives d'administradors.

  Per determinar si un usuari és administrador o no, depén del rol que tingui assignat l'usuari.

  App Engine proporciona tres nivells d'accés a la Consola d'Administració (Figura 2.21):

Rols	**Permissos**
Observador	Pot veure la Consola d'Administració per una aplicació.
Desenvolupador	Pot veure i editar una aplicació a través de la Consola d'Administrador.
Propietari	Pot veure, editar i eliminar una aplicació a través de la Consola d'Administrador. Pot invitar usuaris i canviar els rols.

Un usuari de l'aplicació Code2.0 pot accedir a l'apartat d'Inici i a l'apartat d'Activitats.

L'apartat d'inici permet:

- La visualització d'anuncis/notícies d'ús general pels alumnes.

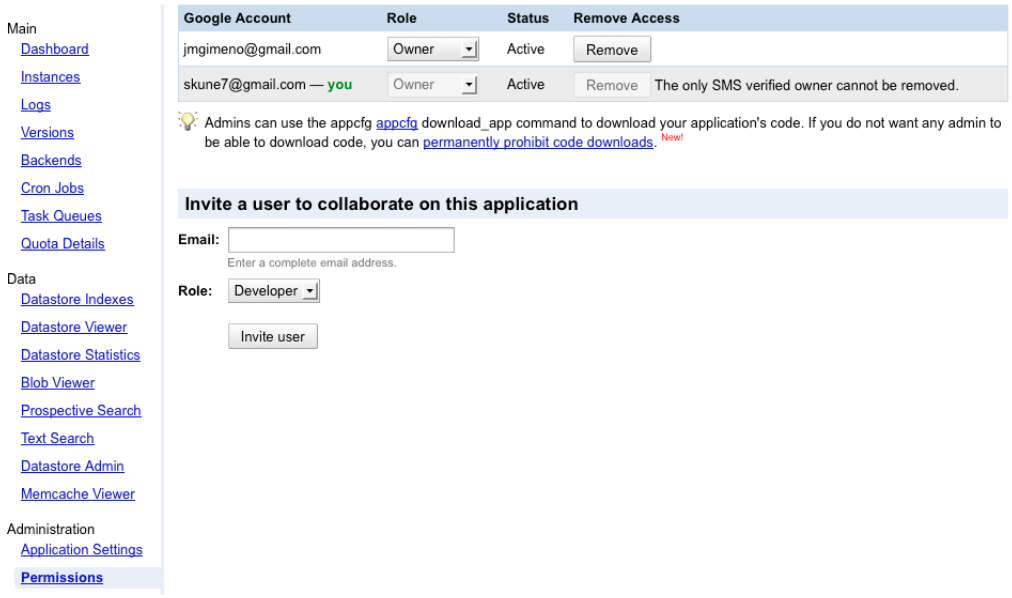

Figura 2.21: Rols

L'apartat d'Activitats permet:

- La visualització de la informació i l'estat d'un anunci.
- Entregar activitats.
- Navegar pels fitxers d'una entrega i visualitzar-los.
- Anotar comentaris a un fitxer.

Per accedir a l'apartat d'Administració (Figura 2.22) és necessari registrar-se a Google Accounts (Figura 2.23) per saber si l'usuari és Administrador, ja que és un apartat restringit per usuaris que no ho són.

Aquest apartat permet:

- Afegir, modificar i eliminar anuncis.
- Afegir, modificar i eliminar assignatures i activitats.
- Configurar alguns paràmetres de l'aplicació web.

Quan un usuari es registra a Google Accounts, el sistema de comptes de Google on està ubicada l'aplicació Code2.0, sol·licita una autorització per accedir al compte de Google amb el qual l'usuari s'ha registrat tal i com es mostra a la figura 2.24.

Un cop s'accepta l'autorització, l'usuari pot ser o no Administrador de l'aplicació.

1. Si l'usuari no és Administrador de l'aplicació, apareix el següent missatge: "El vostre usuari no està autoritzat a realitzar aquesta acció" i no pot accedir a l'apartat d'Administració, tal i com es mostra a la figura 2.25.

Figura 2.22: Accés restringit

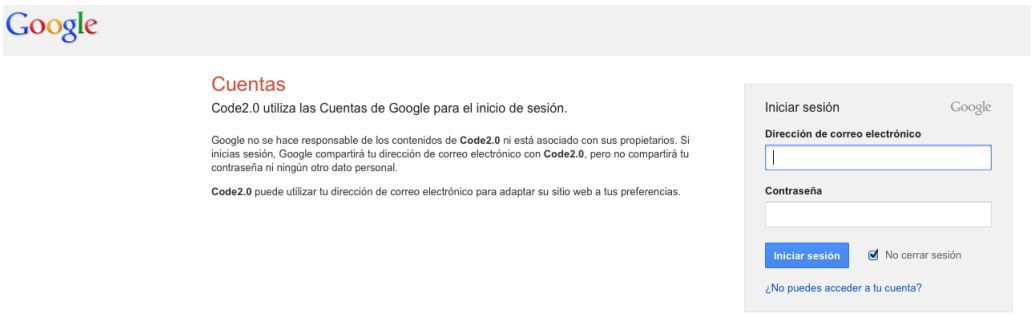

Figura 2.23: Google Accounts

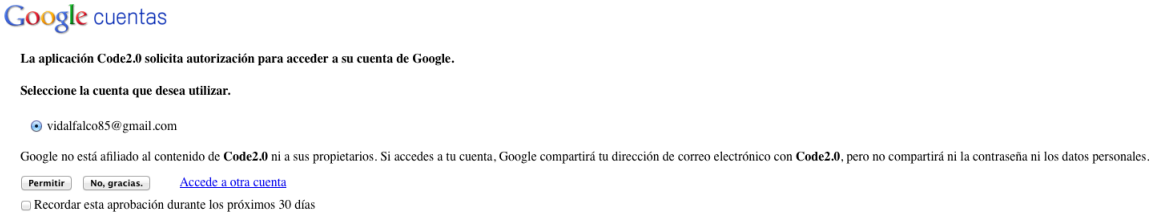

Figura 2.24: Autorització Google Accounts

Figura 2.25: Usuari no autoritzat

Figura 2.26: Usuari Administrador

2. Si l'usuari és Administrador de l'aplicació, mostra els tres subapartats que formen l'apartat d'Administració: Anuncis, Assignatures/Activitats i Configuració.

2.6 Desplegament de l'aplicació a un servidor de Google

Per poder desplegar una aplicació a un servidor de Google, s'han de realitzar varies configuracions a nivell de fitxers. A continuació es detallen les configuracions a realitzar.

- Configurar el fitxer **appengine-web.xml**

  Una aplicació App Engine ha de tenir un fitxer al directori WEB-INF/ anomenat appengine-web.xml. Es tracta d'un fitxer de configuració XML on s'hi especifica l'ID registrat de l'aplicació i un identificador de versió.

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>finalproject-code2</application>
  <version>2</version>
</appengine-web-app>
```

  L'element <*application*> conté l'ID de l'aplicació. Es tracta de l'ID que es registra al crear l'aplicació. Quan es puja una aplicació, AppCfg obté l'ID de l'aplicació a partir d'aquest fitxer.

  L'element <*version*> conté l'dentificador de versió de la última versió del codi de l'aplicació. L'identificador de versió pot estar format per lletres, dígits i guions. AppCfg utilitza aquest identificador de versió quan puja l'aplicació i indica a App Engine que creï una nova versió de l'aplicació amb l'identificador proporcionat o que substitueixi la versió amb aquest identificador si ja n'hi existeix una.

- Configurar el plugin **maven-gae-plugin** al fitxer pom.xml

  El plugin maven-gae-plugin ofereix suport a projectes de Google App Engine. Bàsicament tracta de cobrir totes les operacions bàsiques proporcionades pel SDK de Google App Engine, com l'execució d'aplicacions a nivell local, el desplegament d'aplicacions en el núvol de Google, recuperacions de la producció de registres, entre d'altres.

  Els objectius més comuns del plugin són:

  - **gae:run**. Permet executar aplicacions a nivell local.
  - **gae:deploy**. Permet desplegar aplicacions en el núvol de Google.

```
<plugin>
  <groupId>net.kindleit</groupId>
  <artifactId>maven-gae-plugin</artifactId>
  <version>0.9.2</version>
  <dependencies>
    <dependency>
      <groupId>net.kindleit</groupId>
      <artifactId>gae-runtime</artifactId>
      <version>${gae.version}</version>
      <type>pom</type>
    </dependency>
  </dependencies>
</plugin>
```

Un cop realitzades les configuracions esmentades, ja es pot desplegar l'aplicació al servidor de Google.

2.6.1 Restriccions dels comptes gratuïts de GAE

Un cop desplegada l'aplicació web a la infraestructura de Google, GAE ofereix una sèrie de serveis i recursos, dels quals es pot disfrutar fins a un límit predeterminat amb un compte gratuït. Alguns dels recursos permeten sobrepassar el límit gratuït habilitant la quota de facturació.

A continuació es mostra una taula amb les restriccions més importants:

Recurs	**Restriccions per l'Aplicació**
Número d'aplicacions	10 per usuari (compte Google)
Temps d'execució d'una tasca individual	30 segons
Peticions	1.300.000 uds/dia
Ample de banda (sortida)	10 GB/dia
Ample de banda (entrada)	10 GB/dia
Temps de CPU	6.5 hores CPU/dia
Dades d'emmagatzemament	1 GB
Dades enviades	12 GB
Dades rebudes	115 GB
Tasques	100.000 tasques/dia
Desplegaments	1.000 uds/dia

Capítol 3

Conclusions i treball futur

Les primeres conclusions que puc extreure del projecte Code2.0, es que s'han aconseguit realitzar tots els objectius que s'havien establert a la secció 1.5.

Per una banda, tal i com s'ha comentat a la secció 32.2, la integració de Maven al projecte Code2.0 ha facilitat la gestió del projecte ja que a través del fitxer Pom.xml, s'hi descriu tota la informació referent al projecte i els detalls de configuració utilitzats per Maven per construir l'aplicació com per exemple, el projecte a construir, les seves dependències, els components externs i s'especifica l'ordre de construcció dels elements.

Per altra banda, l'actualització de Java a la última versió existent, Java 7, no ha estat possible ja que Google App Engine actualment no suporta aquesta versió. El motiu es que quan l'aplicació Java s'executa a App Engine, ho fa a través de la màquina virtual de Java (JVM) 6 i de les biblioteques estàndards, per tant, Google App Engine recomana utilitzar Java 6 per compilar i provar l'aplicació. Així doncs, es va decidir actualitzar de la versió Java 4 a Java 6.

En canvi, l'actualització de Google Web Toolkit s'ha pogut realitzar de la versió 1.4 a la versió 2.4 amb èxit. S'ha actualitzat totes les classes que durant els canvis de versions s'han quedat obsoletes.

Per finalitzar, l'adaptació de la plataforma Google App Engine ha estat el gran repte del projecte ja que s'ha tingut de modificar quasibé tot el codi inicial per tal d'apartar-lo als requeriments de la plataforma. $\int$ L'adaptació s'ha realitzat satisfactoriament i actualment l'aplicació Code2.0 està ubicada a un servidor de Google utilitzant un domini gratuït: http://finalproject-code2.appspot.com.

Algunes línies de treball futur que es desprenen d'aquest projecte són les següents:

- Internacionalització

  Suport a múltiples idiomes.

- Optmitzar la seguretat de l'aplicació

  Caldria fer un estudi de seguretat per evitar algunes situacions compromeses: evitar escada de permisos, inserció de codi en consultes SQL, etc.

- Acceptar més tipus d'arxius

Permetre la visualització d'arxius .pdf, documents de text, fulles de càlcul, etc. de forma on-line. També s'hauria de donar suport a més tipus d'arxius comprimits com .rar, .jar, .gzip, entre d'altres.

- Informes de correcció

  Permetre a l'ususari descarregar un informe on apareguin tots els comentaris que s'han realitzar en l'activitat, facilitant la revisió d'aquesta al tenir-ho a mà.

- Analitzar el rendiment de l'aplicació i optimitza'l

  A través de la Consola d'Administració es pot fer un estudi del rendiment per poder-lo millorar.

- Actualitzar a la nova versió les llibreries de Google Web Toolkit i Google App Engine.

  Durant el desenvolupament de l'aplicació han sortit les noves versions de les llibreries (Google Web Toolkit versió 2.5.0 i Google App Engine 1.8.1), però com que els canvis no aportaven millores al desenvolupament de l'aplicació, es va mantenir la versió (Google Web Toolkit versió 2.4 i Google App Engine versió 1.7).

- Implementar el servei de correu que proporciona Google App Engine.

  Les aplicacions poden enviar missatges de correu electrònic mitjançamt el servei de correu. Aquest servei utilitza la infraestructura de Google per enviar missatges de correu electrònic.

Annex I: Manual d'usuari

Aquest manual permetrà aprendre a utilitzar totes les funcionalitats de l'aplicació Code2.0 tant a nivell d'alumne com de professor.

Per accedir a l'aplicació web que hi ha desplegada actualment, s'ha d'introduir la URL *http://finalproject-code2.appspot.com,* tot i que altres instal·lacions poden tenir URLs diferents.

Introducció

L'aplicació Code2.0 consta de 3 apartats ben diferenciats:

- **Inici**: Serà el tauler d'anuncis o notícies, i la pàgina inicial.
- **Activitats**: Es l'apartat on, seleccionant una assignatura, es poden visualitzar les dades d'entrega d'una activitat, els terminis d'entrega, entre d'altra.
- **Administració**: Serà el panell d'administració, des d'on es configuraran i mantindran diferents aspectes de l'aplicació.

Què pot realitzar un Alumne?

En aquesta secció s'explica a quins apartats té accés un alumne i quines funcionalitats pot realitzar en cada un dels apartats.

Inici

Aquest apartat serveix per informar als alumnes de qualsevol tema relacionat amb l'aplicació i els continguts que l'administrador desitgi: Anuncis, Notícies, Novetats, etc. Es visualitzen en ordre cronològic mostrant els més recents primer (Figura 3.1).

Figura 3.1: Pantalla d'Inici

L'enllaç *"Sobre l'aplicació..."* mostra una reduïda descripció sobre l'aplicació Code2.0 (Figura 3.2).

Figura 3.2: Sobre l'aplicació

Activitats

L'apartat d'Activitats permet:

- Veure la informació i l'estat d'una activitat.
- Entregar una activitat.
- Navegar pels fitxers d'una entrega i visualitzar-los.
- Anotar comentaris als fitxers.

Quan un alumne entra a l'apartat d'Activitats, ha de seleccionar l'assignatura que desitja per tal de visualitzar les activitats vinculades (Figura 3.3).

Al seleccionar una assignatura, es mostra totes les seves activitats juntament amb la seva informació complementària com per exemple, el seu estat (Obert o Tancat) i la data límit d'entrega (Figura 3.4).

Figura 3.3: Llista d'assignatures

Figura 3.4: Llista d'activitats d'una assignatura

Si l'alumne prem sobre el títol d'una activitat, pot visualitzar la informació referent a l'activitat, com per exemple, el títol de l'activitat, el seu estat (Obert o Tancat), la data i hora límit, una breu descripció i un fitxer (en el cas que s'hagi adjuntat) (Figura 3.5).

Figura 3.5: Informació sobre l'activitat

L'Alumne, a l'hora de realitzar una entrega, es pot trobar amb dues situacions:

1. Si l'estat de l'activitat és "*Obert*", l'alumne pot fer l'enviament de l'activitat seleccionant l'opció "*Enviar Activitat*". Al seleccionar aquesta opció, s'obra una finestra emergent on s'ha d'introduir, el nom de l'alumne i el fitxer comprimit amb l'extensió .zip que contindrà varis fitxers amb el contingut de l'entrega.

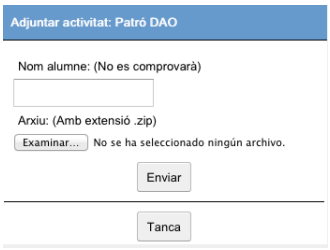

Figura 3.6: Enviar activitat

2. En el cas de que l'estat de l'activitat sigui "*Tancat*", no es pot realitzar l'enviament de l'activitat i es mostra el següent missatge: "*Fora del termini establert: No es pot enviar l'arxiu*".

Un cop l'alumne ha realitzat l'entrega d'una activitat, pot comprovar si l'entrega s'ha efectuat correctament fent un clic sobre la icona que simbolitza una carpeta, al costat del nom de

l'activitat, on es mostrarà el llistat d'entregues realitzades visualitzant el títol de l'entrega, que corresponent al nom del fitxer que ha enviat l'alumne, i el seu nom. tal i com es mostra a la figura 3.7.

Figura 3.7: Llista d'entregues d'una activitat

Per visualitzar el contingut d'una entrega, es pot fer un clic sobre el títol de l'entrega i es mostrarà la llista dels fitxers que conté amb el nom i la mida de cadascú (Figura 3.8).

Figura 3.8: Llista d'arxius d'una entrega

Si l'alumne desitja veure el contingut del fitxer per visualitzar els comentaris que ha realitzat el professor, fent un clic sobre el nom d'un fitxer de text, veurà el contingut d'aquest tal i com es pot veure a la figura 3.9).

Figura 3.9: Contingut d'un fitxer

Què pot realitzar un Professor?

Els professors tenen accés a tots els apartats de l'aplicació Code2.0, es a dir, a l'Inici, a les Activitats i a l'Administració.

Tenir accés a tots els apartats de l'aplicació no significa que totes les funcionalitats que hi ha siguin pel professor. Fins al moment, s'ha explicat què pot realitzar un alumne a l'aplicació Code2.0. A continuació s'explica quines són les funcionalitats que podrà realitzar un professor.

Inici/Activitats

El professor visualitzarà els anuncis que ell mateix pot crear des de l'apartat d'Administració un cop registrat a l'aplicació. A part de veure els anuncis, també podrà accedir a l'apartat d'Activitats. En aquest apartat podrà veure les entregues que els alumnes han realitzat per una assignatura en concret i visualitzar el contingut dels fitxers de text.

En aquests fitxers de text, el professor podrà afegir, modificar o eliminar comentaris. Per realitzar alguna d'aquestes tres accions, s'ha de prémer el número de la línia i a continuació mostra un desplegable que, permetrà escriure un comentari com si d'un editor de text es tractés. L'editor que ens mostra, es pot veure a la figura 3.10).

Totes les opcions que mostra l'editor de text són opcionals (el color, la font, la mida, etc.) exceptuant el camp Tipus, que permet diferenciar un comentari. Aquest camp conté tres opcions que representen l'estat del comentari:

Figura 3.10: Inserir comentari

- **Positiu**: Color verd
- **Neutral**: Color blau
- **Negatiu**: Color vermell

Per exemple, si afegim un comentari en una de les línies, el número de la línia canvia de color a vermell, i sota, s'afegeix el comentari amb una icona de color verd, blau o vermell, segons el tipus de comentari que s'elegeix. Es pot veure un exemple a la figura 3.11.

Administració

Per entrar a l'apartat d'Administració, un professor ha d'autenticar-se introduint el seu correu electrònic, un compte de Google, i la seva contrasenya (Figures 3.12 i 3.13 respectivament).

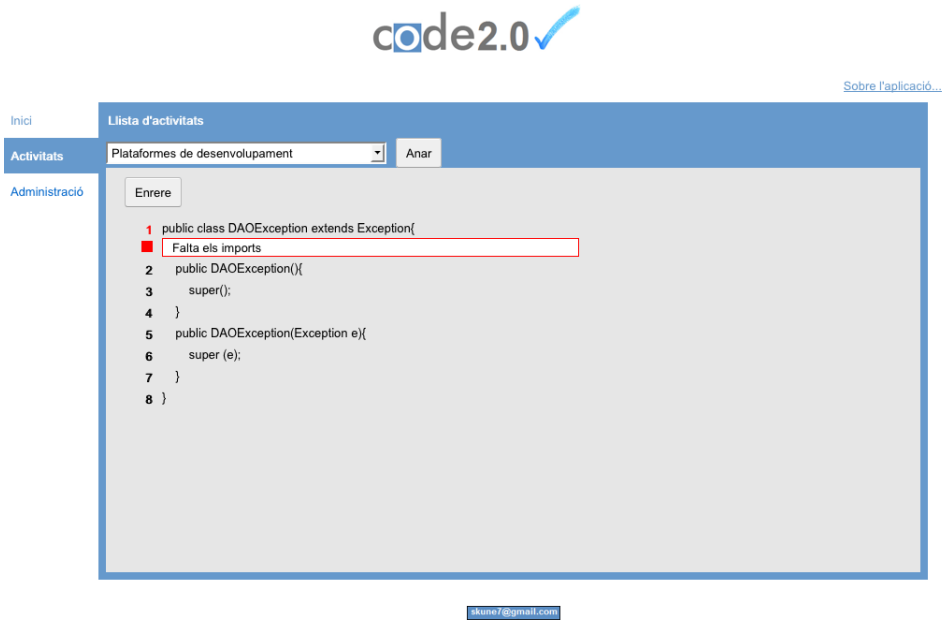

Figura 3.11: Exemple de comentari negatiu

Figura 3.12: Autenticació d'un professor

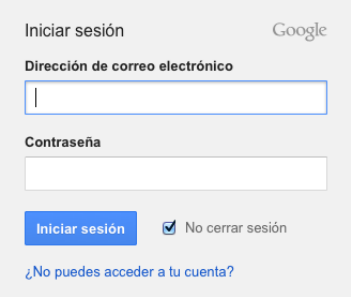

Figura 3.13: Compte de Google

Si les dades d'autenticació són correctes, el sistema el redireccionarà a la pantalla d'inici per a que pugui entrar a l'apartat d'Administració. En cas contrari, li sortirà el següent avís: "L'usuari no té accés per accedir a aquest apartat".

Un cop registrat, un professor pot:

- Afegir, modificar i eliminar anuncis.
- Afegir, modificar i eliminar assignatures i activitats.

L'administració es divideix en dos subapartats: Anuncis i Assignatures/Activitats.

1. **Anuncis**

   En aquest subapartat es creen els anuncis, s'editen els que ja existeixen i es poden eliminar els que siguin obsolets. (Figura 3.14).

Figura 3.14: Administració: Anuncis

Quan s'edita un anunci, es pot modificar el títol i el text però no es pot modificar la data. La data s'actualitzarà en el moment que el professor premi el botó de "Modificar". En aquest moment, a l'anunci s'actualitzarà la data que hi havia amb la data en que s'ha fet la modificació, tal i com es mostra a la figura 3.15).

Figura 3.15: Editar anunci

Quan s'elimina un anunci, l'aplicació demana una confirmació tal i com es mostra a la següent figura 3.16).

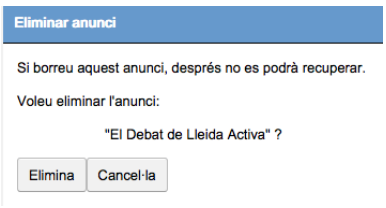

Figura 3.16: Eliminar anunci

2. **Assignatures/Activitats**

   Des d'aquest subapartat es poden administrar les assignatures i les seves respectives activitats.

   Es diferencien dues parts:

   (a) **Creació de les assignatures.**

   Està format per un camp de text on es dóna d'alta les assignatures, escrivint el nom i prement el botó de "Crea" (Figura 3.17).

   Un cop creada una assignatura, el professor té les opcions de canviar el nom o eliminar-la. S'ha de tenir en compte, que a l'eliminar un assignatura (amb prèvia confirmació), s'eliminaran totes les activitats que conté aquesta, juntament amb les entregues (Figura 3.18).

   (b) **Creació de les activitats que pertanyen a una assignatura.**

   Si es selecciona la icona de la carpeta que està al costat del nom de l'assignatura, és llistarà totes les activitats que conté juntament amb l'opció de crear-n'hi una de nova tal i com es mostra a a figura 3.19).

   Quan es crea una activitat, s'ha d'omplir un formulari amb les següents dades (Figura 3.20):

   - Nom de l'activitat
   - Data límit d'entrega

Figura 3.17: Crear assignatura

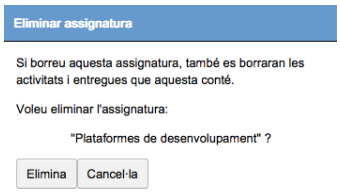

Figura 3.18: Eliminar assignatura

- Hora límit d'entrega
- Una breu descripció
- Un fitxer (Opcional). Està pensat per si el professor vol adjuntar l'enunciat o altres fitxers d'interès.

Si es desitja modificar una activitat, es poden modificar tots els camps comentats anteriorment menys el camp fitxer que no es podrà modificar el fitxer assignat o en el cas que no s'hagi adjuntat cap, tampoc es podrà fer llavors.

Figura 3.19: Llistat d'activitats

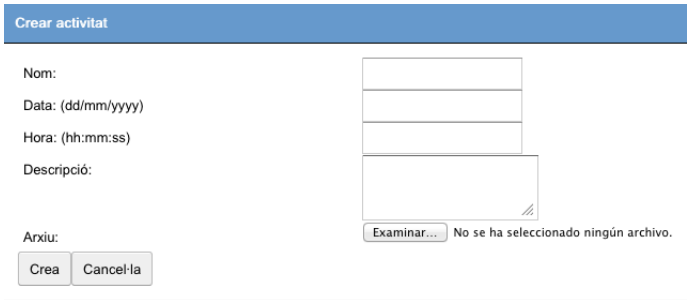

Figura 3.20: Creació d'una activitat

Annex II: Desplegament de l'aplicació

Hi ha dos formes diferents per realitzar el desplegament d'una aplicació:

1. A nivell local.
2. A un servidor de Google.

A nivell local

Google App Engine inclou un servidor web per realitzar proves de l'aplicació a nivell local. Aquest servidor simula un entorn d'execució Java incluint tots els serveis que aporta.

A l'utilitzar l'Eclipse i el complement de Google, es pot executar el servidor web en el seu propi depurador.

Per realitzar una execució amb una configuració específica, s'ha de prémer el botó dret damunt del projecte web i seleccionar l'opció **Run as > Maven buid**, tal i com es mostra a la figura 3.21.

En aquest menú que apareix un cop seleccionada l'opció **Maven build**, figura 3.22, s'ha d'introduir l'objectiu que volem executar en el camp **Goal**, en aquest cas **mvn gae:run**.

En aquest punt, Eclipse crea el projecte i el servidor s'inicia automàticament. Si el servidor s'inicia correctament, apareix varis missatges a la consola sobre el desplegament, com per exemple:

- INFO: The admin console is running at http://localhost:8080
- INFO: The admin console is running at http://localhost:8080/_ah/admin

Si s'accedeix a la primera URL, mostra la pantalla d'inici de l'aplicació Code2.0 (Figura 3.23). En el cas d'introduir la segona URL en el navegador, es mostra la part d'administració de l'aplicació, es a dir, es pot visualitzar les taules de la base de dades amb les seves dades introduides, els logs, configuracions, etc (Figura 3.24).

Amb l'Eclipse, es pot deixar el servidor executant-se mentre es realitzen diferents canvis en el codi font, fitxers estàtics, el fitxer appengine-web.xm, etc. Al guardar els canvis, Eclipse compila la classe de forma automàtica i, a continuació, intenta afegir-la al servidor web que està en execució, i simplement recargant la pàgina del navegador es pot provar la nova versió del codi. En canvi, si es realitza modificacions al fitxer web.xml o altres fitxers de configuració, s'ha d'aturar el servidor i tornar-lo a iniciar per a que s'apliquin els canvis.

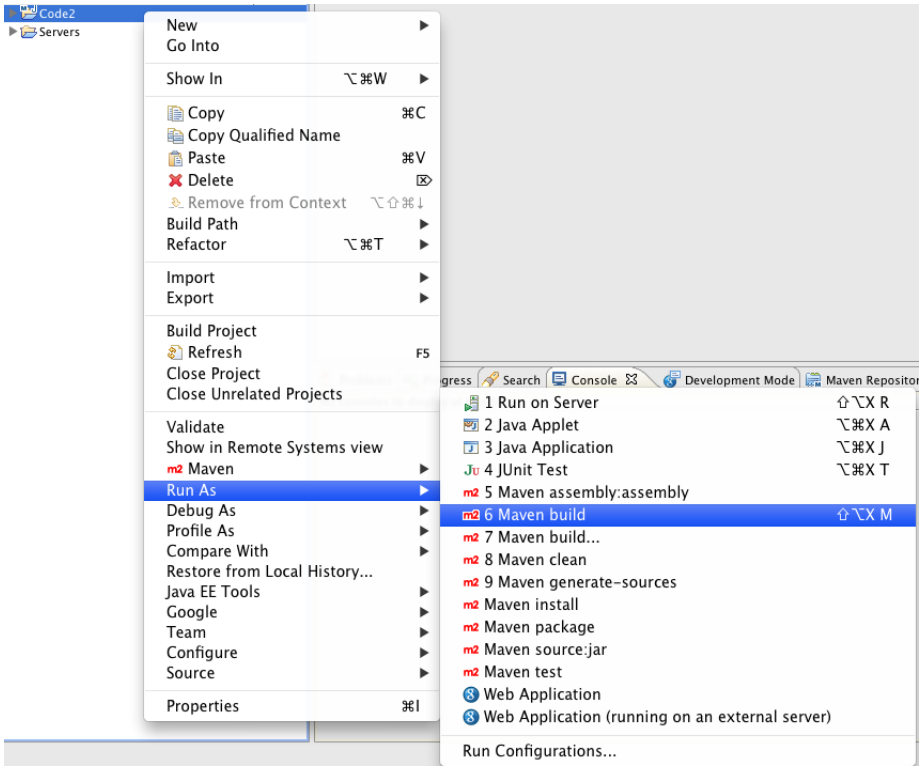

Figura 3.21: Run as > Maven build

A un servidor de Google

Per poder desplegar una aplicació realitzada amb Maven i Google App Engine la comanda que s'ha d'utilitzar es: **mvn gae:deploy** tal i com es mostra en la figura 3.25. Per poder executar la comanda, ens em de situar al directori de l'aplicació on hi ha el fitxer pom.xml.

Un cop introduida la comanda a la consola, es comencen a descarregar tots els plugins configurats en el fitxer de configuració, pom.xml.

En un punt del desplegament on ja s'ha descarregat tots els plugins, es sol·licita introduir el correu electrònic i la contrasenya registrada en Google App Engine per poder continuar (Figura 3.26).

A la consola es va veient el progrés del desplegament i un cop finalitzat, apareix la informació del temps que ha tardat en fer el desplegament i si hi ha hagut algun error, tal i com es mostra a la figura 3.27.

Arribat a aquest punt, l'aplicació ja està ubicada en un servidor de Google i disponible a la següent direcció per poder-hi accedir: *http://finalproject-code2.appspot.com* (Figura 3.28).

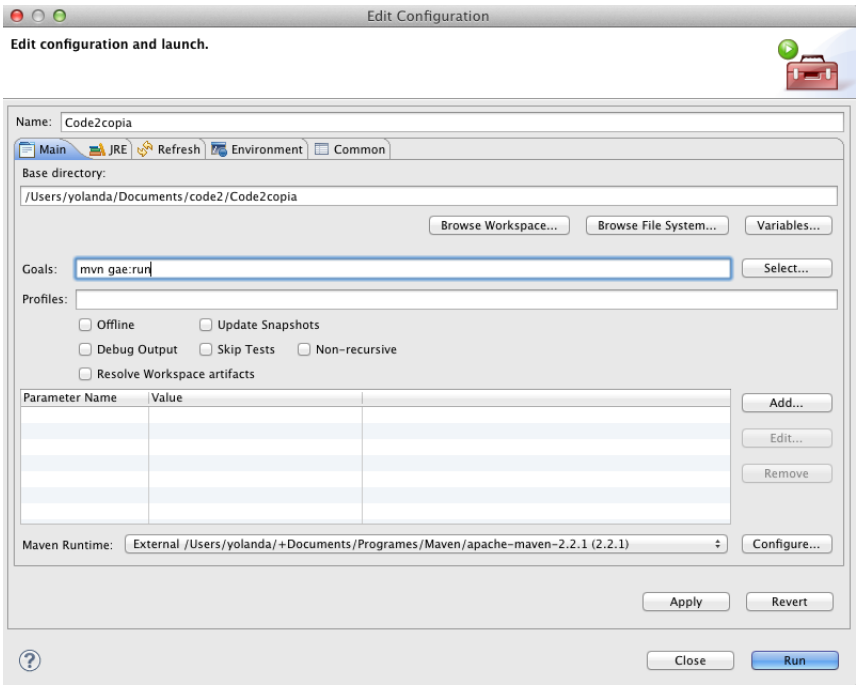

Figura 3.22: Edit configurations

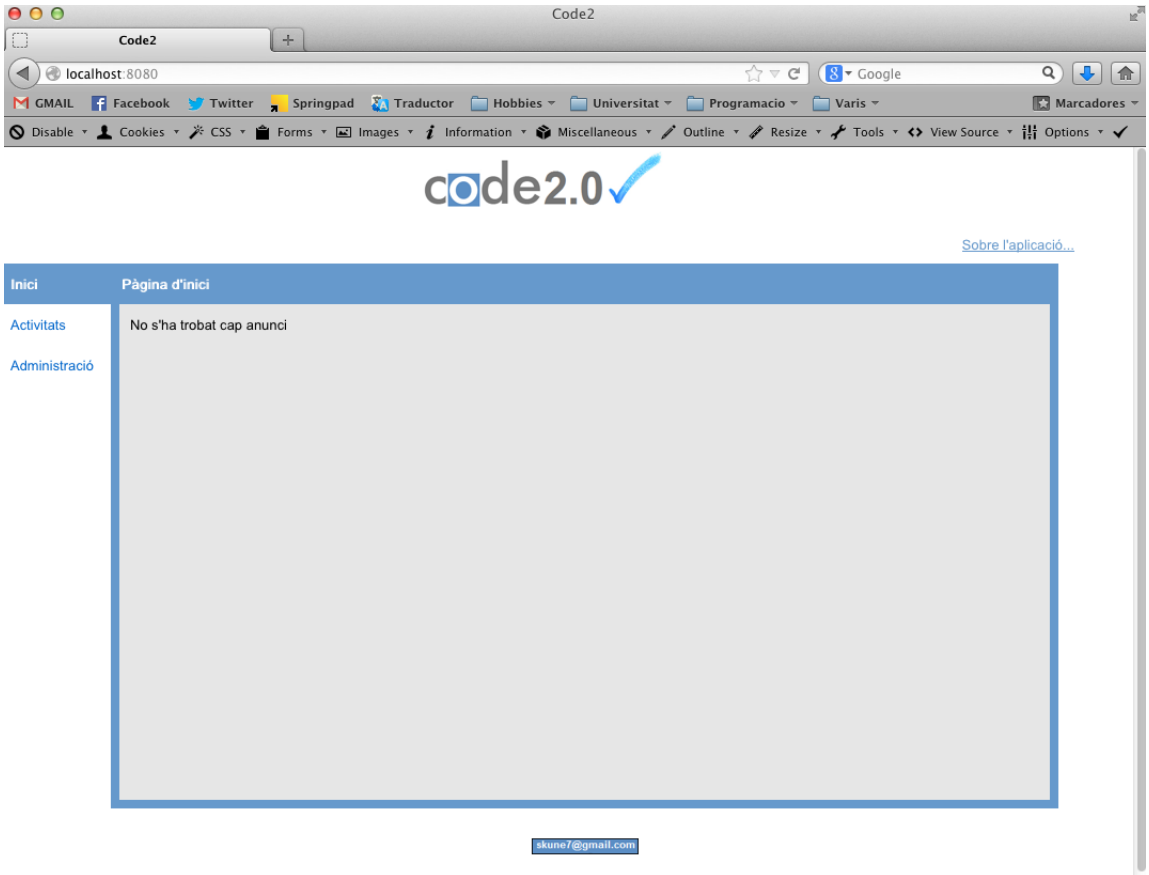

Figura 3.23: http://localhost:8080

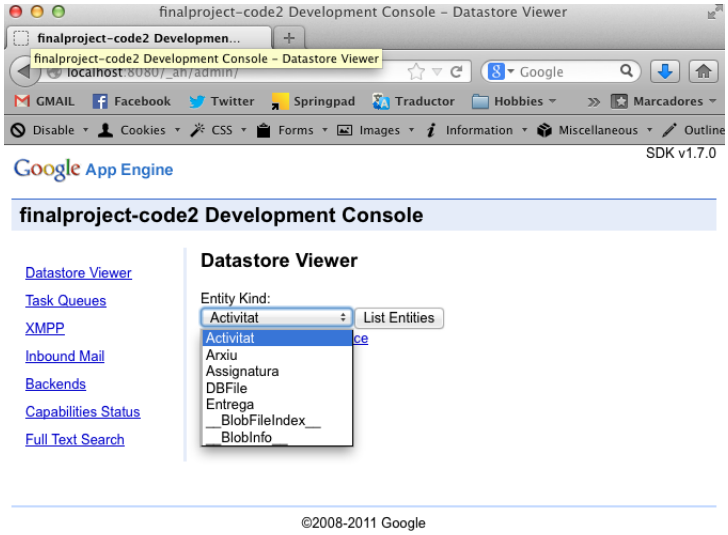

Figura 3.24: http://localhost:8080/_ah/admin

```
[yolanda@MacBook-Pro-de-Yolanda-Vidal-Falco:Code2copia]$ mvn gae:deploy
```

Figura 3.25: Comanda de desplegament d'una aplicació

```
Beginning server interaction for finalproject-code2...
Email: skune7@gmail.com
Password for skune7@gmail.com:
```

Figura 3.26: Identificació a Google App Engine

```
Reading application configuration data...
jun 22, 2013 1:23:42 PM com.google.apphosting.utils.config.AppEngineWebXmlReader readAppEngineWebXml
Información: Successfully processed /Users/yolanda/Documents/code2/Code2copia/target/Code2-1.0-SNAPSHOT/WEB-INF/appengine-web.xml
jun 22, 2013 1:23:42 PM com.google.apphosting.utils.config.AbstractConfigXmlReader readConfigXml
Información: Successfully processed /Users/yolanda/Documents/code2/Code2copia/target/Code2-1.0-SNAPSHOT/WEB-INF/web.xml
jun 22, 2013 1:23:42 PM com.google.apphosting.utils.config.IndexesXmlReader readConfigXml
Información: Successfully processed /Users/yolanda/Documents/code2/Code2copia/target/Code2-1.0-SNAPSHOT/WEB-INF/appengine-generated/datastore-indexes
-auto.xml
Beginning server interaction for finalproject-code2...
Email: skune7@gmail.com
Password for skune7@gmail.com:
0% Created staging directory at: '/var/folders/_z/bmbmhd3s26b77jvsj_p1djkh0000gn/T/appcfg5069230736451127769.tmp'
5% Scanning for jsp files.
20% Scanning files on local disk.
25% Scanned 250 files.
28% Initiating update.
31% Cloning 77 static files.
33% Cloning 282 application files.
34% Cloned 100 files.
35% Cloned 200 files.
40% Uploading 2 files.
52% Uploaded 1 files.
61% Uploaded 2 files.
68% Initializing precompilation...
73% Sending batch containing 2 file(s) totaling 6KB.
90% Deploying new version.
95% Will check again in 1 seconds.
98% Closing update: new version is ready to start serving.
99% Uploading index definitions.
Update completed successfully.
Success.
Cleaning up temporary files...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1:27.072s
[INFO] Finished at: Sat Jun 22 13:24:46 CEST 2013
[INFO] Final Memory: 13M/250M
[INFO] ------------------------------------------------------------------------
```

Figura 3.27: Log del desplegament de l'aplicació

Figura 3.28: Aplicació Code2

Bibliografia

[1] Bazaar [online]. Available from: http://bazaar.canonical.com/en/ [cited Juliol 2013].

[2] Git [online]. Available from: http://git-scm.com/ [cited Juliol 2013].

[3] Google app engine [online]. Available from: https://developers.google.com/appengine/?hl=es [cited Juliol 2013].

[4] Google web toolkit [online]. Available from: https://developers.google.com/web-toolkit/ [cited Juiol 2013].

[5] Google web toolkit api references [online]. Available from: http://www.gwtproject.org/javadoc/latest/ [cited Juliol 2013].

[6] Java [online]. Available from: http://www.clubdesarrolladores.com/articulos/mostrar/38-java-su-historia-ediciones-versiones-y-caracteristicas\-como-plataforma-y-lenguaje-de-programacion/6 [cited Juliol 2013].

[7] Jdo [online]. Available from: http://db.apache.org/jdo/pm.html [cited Juliol 2013].

[8] Json [online]. Available from: http://www.json.org/ [cited Juliol 2013].

[9] Maven [online]. Available from: http://maven.apache.org/ [cited Juliol 2013].

[10] Bryan O'Sullivan. *Distributed revision control with Mercurial*. 2006,2007.

[11] Celia Fontanillo Fontanillo y Antonio Garrote Hernandez. *Tipos Genericos en Java*. Departamento de Informatica y Automatica - Universidad de Salamanca, 2003.